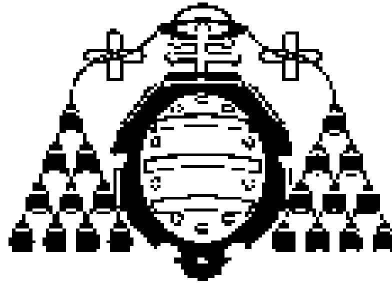
UNIVERSIDAD DE OVIEDO

Departamento de Informática



TESIS DOCTORAL

SAHARA: ARQUITECTURA DE SEGURIDAD INTEGRAL PARA SISTEMAS DE AGENTES MÓVILES

Presentada por

Jesús Arturo Pérez Díaz

Para la obtención de Doctor en Informática

Dirigida por el

Profesor Doctor D. Juan Manuel Cueva Lovelle

Oviedo, Enero de 2000

INTRODUCCIÓN 11
Organización de la Tesis 12

PARTE I.- PREDECESORES DE LOS AGENTES MÓVILES: 14

OBJETOS DISTRIBUIDOS 14

CAPÍTULO I 15
LA SIGUIENTE REVOLUCION CLIENTE/SERVIDOR 15
1.1 REQUERIMIENTOS DE UN SOFTWARE CLIENTE/ SERVIDOR 15
1.2 CLIENTE/SERVIDOR CON BASES DE DATOS SQL: 16
1.3 CLIENTE/SERVIDOR CON MONITORES TP: 16
1.3 CLIENTE/SERVIDOR CON GROUPWARE: 17
1.4 CLIENTE/SERVIDOR CON OBJETOS DISTRIBUIDOS 18
CAPÍTULO II 19
DE OBJETOS DISTRIBUIDOS A COMPONENTES INTELIGENTES 19
2.1 OBJETOS DISTRIBUIDOS 19
2.2 COMPONENTES 20
2.3 SUPERCOMPONENTES 20
CAPÍTULO III 22
CORBA (COMMON OBJECT REQUEST BROKER ARQUITECTURE) 2.0 22
3.1 EL BUS DE OBJETOS ORB(Object Request Broker) 23
3.1.1 La anatomía de un ORB CORBA 2.0 23
3.1.2 Invocación de métodos CORBA. Estáticos contra Dinámicos 25
3.1.3 El lado del servidor de CORBA. El adaptador de objetos 26
3.1.4 Inicialización de CORBA 2.0 ó ¿Cómo un componente encuentra su ORB ? 28
3.1.5 El ORB mundial y la arquitectura Inter-ORB 28
3.2 LOS METADATOS DE CORBA : IDL Y EL REPOSITORIO DE INTERFACES. 28
3.2.1 Las IDLs de CORBA y su estructura 29
3.2.2 El repositorio de interfaces 30
3.3 LOS SERVICIOS DE CORBA : NOMBRADO, EVENTOS Y CICLO DE VIDA 31
3.3.1 El servicio de nombrado de objetos de CORBA 32
3.3.2 El servicio negociación (*trader*) de CORBA 32
3.3.3 El servicio de ciclo de vida de objetos de CORBA 33
3.3.4 El servicio de eventos de CORBA 33
3.4 LOS SERVICIOS DE CORBA: TRANSACCIONES Y CONCURRENCIA: 34
3.4.1 El servicio de transacción de objetos de CORBA 35
3.4.2 El servicio de control de concurrencia de CORBA 36
3.5 LOS SERVICIOS DE CORBA : PERSISTENCIA Y BASES DE DATOS DE OBJETOS 37
3.5.1 El servicio de persistencia de objetos de CORBA (POS) 37
3.5.2 Sistemas gestores de bases de datos orientados a objetos (ODBMSs) 39
3.6 LOS SERVICIOS DE CORBA: CONSULTA Y RELACIONES 40
3.6.1 El servicio de consultas de CORBA 41
3.6.2 El servicio de colección de CORBA 42
3.6.3 El servicio de relación de CORBA 42
3.7 LOS SERVICIOS DE CORBA: GESTION DEL SISTEMA Y SEGURIDAD 43
3.7.1 El servicio de exteriorización de CORBA 44
3.7.2 El servicio de licencias de CORBA 45
3.7.3 El servicio de propiedad de objetos de CORBA 45
3.7.4 El servicio de sincronización de objetos de CORBA 46
3.7.5 El servicio de seguridad de objetos de CORBA 46
3.7.6 El servicio de actualización de objetos de CORBA 47
3.8 LOS OBJETOS DE NEGOCIOS (BUSINESS OBJECT) 47
3.8.1 La anatomía de un objeto de negocios 47
3.9 LAS FACILIDADES COMUNES DE CORBA 48
3.9.1 Las facilidades comunes horizontales de CORBA 48
3.9.2 Las facilidades comunes verticales de CORBA 50
CAPÍTULO IV 51
OPENDOC 51
4.1 OPENDOC : EL MODELO DEL OBJETO 51
4.1.1 El modelo de programación de OpenDoc. 52
4.2 OPENDOC: EL MODELO DE DOCUMENTOS COMPUESTOS 53
4.2.1 Creación del entorno de OpenDoc 54
4.2.2 Gestión de la estructura de documentos. 54

4.2.3 Distribución de eventos y arbitraje....55
4.3 OPENDOC: BENTO Y UNIDADES DE ALMACENAMIENTO....*55*
4.3.1 El sistema de almacenamiento de Bento....56
4.3.2 Unidades de almacenamiento de OpenDoc....56
4.4 OPENDOC : TRASFERENCIA DE DATOS UNIFORME....*57*
4.5 OPENDOC : AUTOMATIZACION Y EVENTOS SEMANTICOS....*57*
4.5.1 El nuevo mundo de programación de OpenDoc.....58
4.5.2 El mecanismo de extensión de OpenDoc....58
4.5.3 Los eventos semánticos....59
4.5.4 Programación y automatización....59
CAPÍTULO V....61
COM (COMPONENT OBJECT MODEL)....61
5.1 COM : EL BUS DE OBJETOS DE OLE....*62*
5.1.1 El estilo de interfaces de COM....62
5.1.2 Qué es un objeto COM....63
5.1.3 Qué es un servidor COM....63
5.2 LOS SERVICIOS DE OBJETOS....*64*
5.2.1 Gestión del Ciclo de Vida:....65
5.2.2 Creación de objetos y licencias....65
5.2.3 El servicio de Eventos de COM....65
5.2.4 El estilo de herencia de COM: Agregación y contención....66
5.3 OLE: AUTOMATIZACION, PROGRAMACION Y LIBRERIAS DE TIPOS....*66*
5.3.1 Automatización y servidores de automatización....67
5.3.2 Creación y gestión de información de tipos, Librerías de tipos.....68
5.4 OLE: TRANSFERENCIA DE DATOS UNIFORME....69
5.4.1 El modelo de transferencia de datos de OLE....69
5.4.2 Trasferencias por portapapeles, mover y soltar, y enlazado de datos....70
5.5 OLE: ALMACENAMIENTO ESTRUCTURADO Y MONIKERS....*71*
5.5.1 Almacenamiento estructurado de OLE: ficheros compuestos....71
5.5.2 Objetos persistentes....72
5.5.3 Monikers....72
5.6 OLE: OCXs Y DOCUMENTOS COMPUESTOS....*73*
5.6.1 El modelo de documentos compuestos de OLE....73
5.6.2 Controles personalizados de OLE (OCXs)....74
5.7 COM DISTRIBUIDO (DCOM):....*75*
5.7.1 La arquitectura básica de DCOM:....75
5.7.2 Condiciones Seguridad....77

PARTE II.- LA TECNOLOGÍA DE AGENTES MÓVILES Y SUS NECESIDADES ACTUALES.....78

CAPÍTULO VI....79
AGENTES: INTRODUCCIÓN, CLASIFICACIÓN Y CONCEPTOS BÁSICOS....79
6.1 INTRODUCCION....*79*
6.2 CONCEPTOS BASICOS....*81*
6.2.1 Definición de agente....81
6.2.2 Albedrío, inteligencia y movilidad.....82
6.3 CLASIFICACIÓN DE AGENTES:....*83*
6.3.1 Los agentes en base a su movilidad.....84
6.3.2 Los agentes en cuanto a su característica fundamental.....84
6.3.3 Diversos tipos de agentes....84
6.4 EVOLUCIÓN DE LOS AGENTES:....*85*
6.4.1 Diversas etapas....85
6.4.2 De agentes inteligentes a agentes móviles....86
6.5 Agentes móviles....*87*
6.5.1 El paradigma de Agentes móviles....88
6.5.1.1 Aproximación actual....88
6.5.1.2 Nueva aproximación....89
6.5.1.3 Ventajas de la programación móvil....89
6.5.2 Conceptos de Agentes móviles....89
6.5.2.1 Lugares....90
6.5.2.2 Agentes....90
6.5.2.3 Viajes....90
6.5.2.4 Entrevistas (meetings)....90
6.5.2.5 Conexiones....91

6.5.2.6 Autoridades ... 91
6.5.2.6 Permisos ... 91
6.5.3 La tecnología de agentes móviles ... 91
6.5.3.1 El lenguaje ... 92
6.5.3.2 La máquina o interprete ... 92
6.5.3.3 Protocolos ... 92
CAPÍTULO VII ... 94
EL POR QUÉ Y EL PARA QUÉ DE LOS AGENTES MÓVILES ... 94
7.1 LOS AGENTES MÓVILES Y OTRAS TECNOLOGÍAS ... 94
7.1.1 Objetos distribuidos, software intermedio orientado a mensajes y agentes móviles ... 94
7.1.2 Agentes móviles vs applets ... 95
7.2 VENTAJAS DE LOS AGENTES MÓVILES: SIETE BUENAS RAZONES PARA USARLOS ... 96
7.3 APLICACIONES DE LOS AGENTES MÓVILES ... 98
7.3.1 Un ejemplo de consultas a Sistemas de Información Geográfica (SIG) mediante agentes móviles ... 101
CAPÍTULO VIII ... 104
EL MUNDO DE LOS AGENTES MÓVILES ... 104
8.1 PORTABILIDAD Y SEGURIDAD ... 105
8.1.1 La necesidad de una máquina virtual ... 105
8.2 PLATAFORMAS PARA EL DESARROLLO DE AGENTES MÓVILES ... 106
8.2.1 Telescript ... 106
8.2.2 D'Agents ... 107
8.2.3 Ara: Agentes para acceso remoto ... 107
8.2.4 JAVA ... 108
8.2.4.1 La máquina virtual y portabilidad ... 108
8.2.4.2 Seguridad ... 109
8.2.4.3 Serialización de objetos, RMI y componentes ... 109
8.2.4.4 Multihilo y Multitarea ... 109
8.2.4.5 Todo listo para los sistemas de Agentes Móviles ... 110
8.3 AGENT TCL O D'AGENTS ... 110
8.3.1 Generalidades de D'Agents ... 111
8.3.1.1 Movilidad de agentes ... 111
8.3.1.2 Comunicación entre agentes ... 111
8.3.1.3 Otras características de D'Agents ... 111
8.3.2 Objetivos de la arquitectura de D'Agents ... 111
8.3.3 La arquitectura de D'Agents ... 112
8.3.4 El lenguaje de desarrollo de D'Agents ... 113
8.3.4.1 Comandos para la creación de agentes ... 114
8.3.4.2 Comandos para la migración de agentes ... 114
8.3.4.3 Comandos para la comunicación de agentes ... 114
8.3.5 La seguridad en D'Agents ... 114
8.3.5.1 Autenticación ... 115
8.3.5.2 Dos debilidades del esquema de autenticación. ... 115
8.3.5.3 Autorización y cumplimiento de privilegios ... 116
8.3.5.4 Las deficiencias del esquema de autorización y cumplimiento de privilegios ... 117
8.3.6 Evaluación de D'Agents ... 117
8.3.6.1 Ventajas ... 118
8.3.6.2 Desventajas ... 118
8.4 ARA: AGENTES PARA ACCESO REMOTO ... 118
8.4.1 Generalidades de Ara ... 119
8.4.1.1 Agentes, lenguajes, y el sistema Ara ... 119
8.4.1.2 La vida de un agente ... 119
8.4.1.3 Movilidad de agentes ... 120
8.4.1.4 Acceso al sistema servidor ... 120
8.4.2 La arquitectura de Ara ... 121
8.4.2.1 Procesos y arquitectura interna ... 121
8.4.2.2 El proceso de comunicación ... 122
8.4.2.3 Protección ... 122
8.4.2.4 Salvado y restauración del estado de un proceso ... 123
8.4.2.5 Relación entre el núcleo y los intérpretes ... 123
8.4.3 Características de Ara y conceptos de programación ... 124
8.4.3.1 Comandos para la gestión básica de un agente ... 124
8.4.3.2 Comandos relacionados con el tiempo ... 125
8.4.3.3 El comando de movilidad ... 125
8.4.3.4 Comandos que definen el comportamiento de lugares ... 125
8.4.3.5 Comandos para la gestión de asignaciones ... 125
8.4.4 La seguridad en Ara ... 125

8.4.4.1 Asignaciones para limitar el acceso a recursos. 126
8.4.4.2 Entrada a un lugar 126
8.4.4.3 Problemas abiertos. 127
8.4.5 Evaluación de Ara 127
8.4.5.1 Ventajas 127
8.4.5.2 Desventajas 128
8.5 IBM AGLETS 128
8.5.1 El modelo del objeto Aglet 128
8.5.1.1 Generalidades de la API de los aglets 129
8.5.1.2 Heredando de la clase aglet 130
8.5.1.3 El objeto aglet y su ciclo de vida 132
8.5.1.4 Los eventos de los aglets y el modelo de delegación de eventos 133
8.5.2 Migración de objetos y mensajes con aglets 134
8.5.2.1 Diversos tipos de serialización 134
8.5.2.2 Mensajes con aglets 135
8.5.2.3 Paso de mensajes de forma remota con aglets 135
8.5.3 La arquitectura del sistema 136
8.5.3.1 El protocolo de transferencia de agentes 137
8.5.3.2 El uso de hilos en atp 138
8.5.3.3 Carga de clases 138
8.5.3.4 Movilidad de Clases 139
8.5.3.5 Http tunneling a través de servidores proxy 140
8.5.4 Seguridad en los Aglets 141
8.5.4.1 La base de Datos de Políticas de seguridad 141
8.5.4.2 La clase AgletSecurityManager 142
8.5.5 Construcción de aplicaciones con aglets 143
8.5.5.1 Componentes con solo aglets 143
8.5.5.2 Aplicaciones que incrustan un servidor de Aglets 144
8.5.5.3 Aplicaciones cliente 144
8.5.5.4 Herramientas adicionales del AgletsFramework 144
8.5.6 Evaluación del sistema de los aglets 145
8.5.6.1 Ventajas 145
8.5.6.2 Desventajas 145
8.6 UNA ALTERNATIVA PARA LA COMUNICACIÓN ENTRE AGENTES: KQML 146
8.6.1 Una descripción de KQML 146
8.5.2 Facilitadores y el entorno de agentes parlantes de KQML 148
8.6.3 Una breve evaluación de KQML como lenguaje de comunicación de agentes. 148
8.6.4 Aplicaciones de KQML 149
8.7 FACILIDAD DE INTEROPERABILIDAD DE SISTEMAS DE AGENTES MÓVILES DE CORBA (MASIF): UN INTENTO DE ESTANDARIZACIÓN 150
8.7.1 Interoperabilidad 150
8.7.1.1 ¿Qué se debe estandarizar ahora? 150
8.7.1.2 ¿Qué se debe estandarizar más tarde ? 151
8.7.1.3 Sumario de la interoperabilidad de la MASIF. 151
8.7.2 Conceptos básicos 151
8.7.2.1 Autoridad del agente 151
8.7.2.2 Nombres del agente 151
8.7.2.3 Localización del agente 152
8.7.2.4 Sistema de agentes 152
8.7.2.5 Tipo del sistema de agentes 152
8.7.2.6 Interconexión de sistema de agentes a sistemas de agentes 152
8.7.2.7 Lugar 153
8.7.2.8 Regiones 153
8.7.2.9 interconexión de región a región 153
8.7.2.10 Serialización de serialización 154
8.7.2.11 Código base 154
8.7.2.12 Infraestructura de comunicación 154
8.7.2.13 Localidad 154
8.7.3 Interacción de Agentes 154
8.7.3.1 Creación de agentes remoto 155
8.7.3.2 Transferencia de agentes 155
8.7.3.3 Invocación de métodos del agente 155
8.7.4 Funciones de un sistema de agentes 155
8.7.4.1 Transferencia de un agente 155
8.7.4.2 Creación de un agente 156
8.7.4.3 Provisión de nombres y localizaciones únicas globalmente 157
8.7.4.4 Soportar el concepto de una región 157

8.7.4.5 Encontrar un agente móvil 157
8.7.4.6 Asegurar un entorno seguro para las operaciones de los agentes 157
8.7.4.7 Autenticación 157
8.7.5 IDL de la MASIF 159
8.7.5.1 Nombre, nombres de clase y localización 159
8.7.5.2 Localización 160
8.7.5.3 Servicio de nombrado de OMG de identificadores de autoridad 161
8.7.6 La interfaz MAFAgentSystem 162
8.7.6.1 create_agent() 163
8.7.6.2 fetch class() 163
8.7.6.3 find_nearby_agent_system_of_profile() 163
8.7.6.4 get_agent_status() 163
8.7.6.5 get_agent_system_info() 163
8.7.6.6 get_authinfo() 163
8.7.6.7 get_MAFFinder() 163
8.7.6.8 list_all_agents() 164
8.7.6.9 list_all_agents_of_authority() 164
8.7.6.10 list_all_places() 164
8.7.6.11 receive_agent() 164
8.7.6.12 resume_agent() 164
8.7.6.13 suspend_agent() 164
8.7.6.14 terminate_agent() 164
8.7.6.15 terminate_agent_system() 164
8.7.7 La interfaz MAFFinder 164
8.7.7.1 lookup_agent() 165
8.7.7.2 lookup_agent_system () 165
8.7.7.3 lookup_place() 165
8.7.7.4 register_agent () 166
8.7.7.5 register_agent_system () 166
8.7.7.6 register_place () 166
8.7.7.7 unregister_agent () 166
8.7.7.8 unregister_agent_system () 166
8.7.7.9 unregister_agent () 166
8.8 COMPARACIÓN, CARENCIAS Y TENDENCIAS DE LOS SISTEMAS DE AGENTES MÓVILES 166
8.8.1 Comparación 167
8.8.2 Carencias 168
8.8.2.1 Seguridad 168
8.8.2.2 Interoperabilidad 168
8.8.2.3 Entornos visuales de desarrollo e interfaces gráficas del usuario 169
8.8.2.4 Incorporación a los navegadores 169
8.8.3 Tendencias 169
CAPÍTULO IX 171
INTEGRACIÓN DE LOS AGENTES MOVILES A SISTEMAS ORIENTADOS A OBJETOS 171
9.1 ¿CÓMO INTEGRAR AGENTES MÓVILES EN UN SISTEMA ORIENTADO A OBJETOS? 172
9.2 ¿CÓMO REPRESENTAR AGENTES MÓVILES EN UML? 173
9.3 CLASIFICACIÓN DE PATRONES DE DISEÑO DE AGENTES MÓVILES 175
9.3.2 Patrones de Migración 175
9.3.2 Patrones de Tareas 176
9.3.3 Patrones de interacción 177
9.4 INTERACCIÓN ENTRE AGENTES Y ACCESO A BASES DE DATOS PARA SISTEMAS DE AGENTES MÓVILES BASADOS EN JAVA 179
9.5 INCORPORACIÓN DE AGENTES MÓVILES A SOO CON TECNOLOGIA CORBA 181
CAPÍTULO X 183
REQUERIMIENTOS DE UNA INFRAESTRUCTURA PARA EL DESARROLLO DE AGENTES MÓVILES Y LA NECESIDAD DE UNA ARQUITECTURA DE SEGURIDAD INTEGRAL 183
10.1 REQUERIMIENTOS DE UN SISTEMA DE AGENTES MÓVILES 184
10.1.1 Características deseables del lenguaje de programación de agentes 184
10.1.2 Seguridad 185
10.1.3 Servicios de Agentes 185
10.1.3.1 Movilidad 186
10.1.3.2 Servicio de localización 186
10.1.3.3 Comunicación 187
10.1.4 Interoperabilidad con otros sistemas de agentes 188
10.1.5 Integración al WEB 188
10.1.5.1 Integración a los navegadores 188

10.1.5.2 Agentes dentro de applets 188
10.1.5.3 Páginas amarillas 189
10.1.5 Interfaces gráficas y monitores de agentes 189
10.1.6 Herramientas de desarrollo 189
10.1.6.1 Entornos visuales de desarrollo 190
10.1.6.2 Un depurador y un gestor de excepciones 190
10.2 NECESIDAD DE UNA ARQUITECTURA DE SEGURIDAD INTEGRAL 190
10.2.1 La falta de protección total del servidor de agentes 191
10.2.1.1 El problema de falsificación de agentes y servidores 191
10.2.1.2 Los esquemas inadecuados para la asignación de privilegios 191
10.2.1.3 La mala protección contra el consumo de recursos del sistema 192
10.2.1.4 La carencia de versatilidad de las políticas de seguridad en tiempo de ejecución 193
10.2.2 Carencia de seguridad en la transmisión de agentes 194
10.2.2.1 Espionaje de la información 194
10.2.2.2 Poca fiabilidad en la transmisión de mensajes 194
10.2.3 Falta de protección del agente y sus datos contra ataques de un servidor malicioso 195
10.2.3.1 El problema de alteración código y datos 195
10.2.3.2 El problema de replicas 196

PARTE III .- ARQUITECTURA DE SEGURIDAD INTEGRAL PARA SISTEMAS DE AGENTES MOVILES 198

CAPÍTULO XI 199
SAHARA: ARQUITECTURA DE SEGURIDAD INTEGRAL PARA SISTEMAS DE AGENTES MÓVILES 199
11.1 INTRODUCCION A LA SEGURIDAD EN LOS SISTEMAS DE AGENTES MÓVILES 200
11.1.1 Conceptos básicos 202
11.2 ESQUEMA GLOBAL DE SAHARA EN LA INTERACCIÓN ENTRE SISTEMAS DE AGENTES MÓVILES 203
11.2.1 Objetivos de la arquitectura 203
11.2.2 Autoridades, usuarios y recursos 205
11.2.3 Uso de lugares para definir contextos de ejecución afines 205
11.2.4 Uso de regiones para crear redes confiables 206
11.2.5 Agrupaciones de regiones para crear redes seguras 207
11.2.6 Panorama general del funcionamiento de SAHARA 208
CAPÍTULO XII 211
SAHARA: PROTECCIÓN DEL SISTEMA DE AGENTES CONTRA ATAQUES 211
12.1 AUTENTICACIÓN DE AGENTES DENTRO DE UNA REGIÓN 212
12.2 AUTENTICACIÓN DE AGENTES PROVENIENTES DE REDES NO CONFIABLES 212
12.2.1 Criptografía de llave pública con distribución de llaves públicas 213
12.2.2 Criptografía de llave pública mediante firmas digitales 213
12.2.3 Autoridades de certificación 213
12.2.4 Almacenamiento de las llaves y certificados de cada autoridad 215
12.2.5 Firmado de agentes móviles 216
12.2.6 Proceso de autenticación en SAHARA 217
12.2.7 ¿Qué se logra mediante la autenticación? 218
12.3 AUTORIZACIÓN 218
12.3.1 Uso de asignaciones para restringir el acceso a recursos solicitados por el usuario y controlados por el sistema 219
12.3.2 Autorización de las asignaciones 220
12.3.3 Tipos de permisos para los recursos del sistema 221
12.3.4 Aclaraciones sobre la Implicación de permisos 224
12.3.5 Formato del fichero que define la política de seguridad y la asignación de privilegios 224
12.3.6 Expansión de propiedades en los ficheros de políticas de seguridad 226
12.3.7 Autorización de privilegios para los agentes locales 227
12.3.8 Autorización de privilegios para los agentes que provienen de una misma región 228
12.3.9 Autorización de privilegios para los agentes que provienen de redes no confiables 228
12.3.10 Asignación de Permisos 229
12.3.11 Soporte de múltiples ficheros de seguridad 230
12.3.12 Versatilidad para que los agentes compartan ficheros de forma protegida 230
12.3.12.1 El método share 230
12.4 CUMPLIMIENTO DE PRIVILEGIOS 231
12.4.1 Cumplimiento de los privilegios otorgados en los ficheros de políticas de seguridad del sistema de agentes 232
12.4.2 Resolución de permisos en invocaciones a dominios de protección distintos 234
12.4.3 Cumplimiento de las asignaciones 235

12.4.4 ¿Qué se logra con la Autorización y el Cumplimiento de Privilegios? 236
CAPÍTULO XIII 238
SAHARA: PROTECCION DE LA TRANSMISIÓN DE AGENTES E INFORMACIÓN CONTRA TERCERAS PARTES 238
13.1 PGP o SSL 238
13.2 LAS VENTAJAS DE SSL 239
13.3 DESCRIPCIÓN DEL PROTOCOLO SSL 240
13.3.1 Los estados de sesión y conexión 241
13.3.2 La capa de registro 241
13.3.3 El protocolo de saludo 241
13.3.4 Atributos usados del servidor de agentes de SAHARA 242
13.4 ¿QUÉ SE LOGRA MEDIANTE EL USO DE SSL EN LA TRANSMISIÓN? 243
CAPÍTULO XIV 244
SAHARA: PROTECCIÓN DE AGENTES MÓVILES CONTRA ATAQUES DE SERVIDORES MALICIOSOS 244
14.1 PROTECCIÓN DEL CÓDIGO 244
14.2 PROTECCIÓN DE LOS DATOS 245
14.2.1 Protección de los datos mediante navegación en redes seguras 245
14.2.2 Protección de datos mediante técnicas de duplicación de datos 246
14.2.3 Protección de datos mediante técnicas de duplicación de agentes móviles 247
14.2.4 Protección de datos mediante códigos de autenticación de resultados parciales 248
14.2.5 Protección de datos mediante composición de matrices 250
14.2.6 Protección de datos mediante funciones homomórficas 250
14.3 ¿QUÉ SE LOGRA CON LA PROTECCIÓN DEL CÓDIGO Y DATOS? 251
CAPÍTULO XV 253
SAHARA: AUTOCONTROL DE RECURSOS Y ALCANCES DE LA ARQUITECTURA 253
15.1 AUTOCONTROL DE RECURSOS 253
15.1.1 Autocontrol del espacio en Disco 254
15.1.2 Autocontrol de la memoria RAM 255
15.1.3 Autocontrol de las conexiones a la red 255
15.2 ALCANCES y LIMITACIONES DE LA ARQUITECTURA 256
15.2.1 Alcances de SAHARA 256
15.2.2 Limitaciones de implementación de SAHARA 257

PARTE IV .- EL PROTOTIPO MILENIO 259

CAPÍTULO XVI 260
EL PROTOTIPO MILENIO: UN SISTEMA DE AGENTES MOVILES CON SEGURIDAD INTEGRAL 260
16.1 DEFINICIÓN DEL LENGUAJE 261
16.1.1 Justificación del lenguaje Java 2 SDK (JDK 1.2) 262
16.2 DESCRIPCIÓN GENERAL DEL PROTOTIPO MILENIO 262
16.2.1 El núcleo 264
16.2.2 Componentes de administración del sistema 264
16.2.3 Arquitectura de seguridad SAHARA 265
16.2.4 Diagrama general de clases 266
16.2.4.1 Diagrama general de clases del Núcleo y Sistema de Agentes 266
16.2.4.2 Diagrama general de clases de la Arquitectura de Seguridad 268
CAPÍTULO XVII 271
MILENIO: DESCRIPCIÓN DETALLADA DE LA IMPLEMENTACIÓN DEL NÚCLEO 271
17.1 ELEMENTOS DE UN AGENTE MÓVIL 271
17.2 CREACIÓN E INICIACIÓN DE AGENTES 274
17.3 SERIALIZACIÓN DE AGENTES 275
17.4 MIGRACIÓN DEL AGENTE 276
17.5 TRANSFERENCIA DE CLASES 277
17.5.1 Clases transferidas con el agente 278
17.5.2 Clases requeridas desde el lugar de procedencia del agente 279
17.5.3 Clases del sistema cargadas desde el classpath 279
17.6 PROTOCOLO DE TRANSFERENCIA DE AGENTES 279
CAPÍTULO XVIII 283
MILENIO: DESCRIPCIÓN DE LA IMPLEMENTACIÓN DE LA ARQUITECTURA DE SEGURIDAD SAHARA 283
18.1 SEGURIDAD EN EL ACCESO AL SISTEMA Y EN LA ASIGNACIÓN DE SUS RECURSOS 283

18.2 SEGURIDAD DEL REPOSITORIO DE CLAVES Y DE LOS FICHEROS DE SEGURIDAD285
18.3 EL PROCESO DE FIRMADO DE UN AGENTE....287
18.3.1 Alta de autoridades/usuarios y generación de firmas 287
18.3.2 Preparación del agente para el envío 288
18.3.3 Proceso de firmado 289
18.4 PROCESO DE ENVÍO Y SEGURIDAD EN LA TRANSMISIÓN 289
18.5 PROCESO DE CARGA, AUTENTICACIÓN, ASIGNACIÓN DE PRIVILEGIOS Y DOMINIOS DE PROTECCIÓN....290
18.5.1 Requerimientos previos a la carga del agente....290
18.5.2 El proceso de carga de un agente móvil y el *AgentLoader* 291
18.5.3 Autenticación del agente....292
18.5.4 Asignación de privilegios asociando a dominios de protección existentes....292
18.5.5 Asignación de privilegios creando nuevos Dominios de Protección....293
18.6 VERIFICACIÓN DE ASIGNACIONES Y CUMPLIMIENTO DE PRIVILEGIOS 294
18.6.1 DBResourcesFree y DBResources....295
18.6.2 Verificación de las asignaciones individuales del agente....296
18.6.3 Cumplimiento de privilegios 296
18.7 PERMISOS OTORGADOS POR AGENTES 298
18.8 AUTOCONTROL DEL SISTEMA Y MODIFICACIÓN DE LA POLÍTICA DE SEGURIDAD EN TIEMPO DE EJECUCIÓN 299
18.9 LO QUE NO SE IMPLEMENTO DE SAHARA EN MILENIO 300
CAPÍTULO IXX....301
EL ENTORNO INTEGRADO DE DESARROLLO DE MILENIO....301
19.1 Acceso al sistema....301
19.2 Ventana principal del sistema 302
19.2.1 La barra de Menús....303
19.2.2 La barra de herramientas 303
19.2.3 Area de Agentes Activos....303
19.2.4 Ventanas de salida estándar y de error 303
19.2.5 la barra de estado....304
19.3 Menú Agente....304
19.3.1 Crear nuevo Agente....304
19.3.1.1 Seleccionar Clase....305
19.3.1.2 Seleccionar Itinerario 305
19.3.2 Clonar Agente....306
19.3.3 Información de agente....307
19.4 Menú Ver....308
19.4.1 Barra de herramientas....308
19.4.2 Historial....309
19.4.3 Información del Sistema....309
19.4.4 Usuarios....310
19.4.5 Asignaciones totales 311
19.4.6 Asignaciones disponibles 311
19.5 Menú Ayuda....311

PARTE V .- EVALUACIÓN DEL PROTOTIPO, CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS....313

CAPÍTULO XX 314
APLICACIÓN DE SAHARA A OTROS SISTEMAS DE AGENTES MÓVILES Y TRABAJO RELACIONADO 314
20.1 EL MODELO DE SEGURIDAD PARA LOS AGLETS 315
20.1.1 El lenguaje de autorización....316
20.1.2 Los privilegios....317
20.1.3 La base de datos de políticas del administrador del contexto....317
20.1.4 Preferencias del propietario de los aglet....318
20.1.5 Aplicación de SAHARA al modelo de seguridad de los Aglets....319
20.1.5.1 Gran cantidad de Autoridades 319
20.1.5.2 Perdida de desempeño al utilizar un interprete para el lenguaje de seguridad....320
20.1.5.3 Posibilidad del ataque de agotamiento de recursos....320
20.1.5.4 Política de Seguridad Estática....320
20.2 LA APROXIMACIÓN DE FRITZ HOHL PARA RESOLVER EL PROBLEMA DEL SERVIDOR MALICIOSO....321
20.2.1 Revoltura de código 321
20.2.2 Tiempo de vida limitado de código y datos 322

20.2.3 Análisis de la aproximación y cómo puede ayudar SAHARA al modelo de seguridad de MOLE..322
20.3 APLICACIÓN GENERAL DE SAHARA A OTROS SISTEMAS DE AGENTES MÓVILES....323
20.3.1 Agentes Modulares y Compresión....324
20.3.2 Restricción del Acceso al Sistema....324
20.3.3 Autenticación Múltiple....324
20.3.4 Firmas de los Servidores para Adjudicar Responsabilidad....325
20.3.5 Seguridad en la transmisión mediante SSL....325
20.3.6 Uso del concepto de región para crear redes confiables....325
CAPÍTULO XXI....326
EVALUACIÓN Y AMBITOS DE APLICACIÓN DE MILENIO....326
21.1 EVALUACIÓN Y COMPARACIÓN DE MILENIO CON OTROS SISTEMAS....326
21.1.1 Análisis comparativo de la seguridad de MILENIO contra otros sistemas de agentes móviles....327
21.1.1.1 Las características necesarias que todo sistema debiera tener para ser seguro....328
21.1.1.2 Comparativa de la seguridad de MILENIO contra otros sistemas de Agentes Móviles....329
21.1.2 El Precio de la Seguridad en el Desempeño del Sistema....331
21.2 ÁMBITOS DE APLICACIÓN de MILENIO....332
21.2.1 Aplicaciones donde se paga el acceso individual a un recurso....333
21.2.2 Sistemas de Comercio electrónico....334
21.2.2.1 Verificación y restricción de acceso al mercado electrónico....334
21.2.2.2 Consumación de operaciones....334
21.2.3 Operaciones Bancarias....335
21.2.3.1 Operaciones seguras, programables y autónomas....335
21.2.3.2 Transacciones inmediatas....336
21.2.4 Distribución selectiva y certificada de información....336
21.2.4.1 Distribución de programas y actualización de componentes....336
21.2.4.2 Entrega de correo electrónico certificado y acuse de recibo....336
21.2.5 Administración de Servidores y Balanceo de Tráfico en Grandes Redes....337
21.2.6 Aplicaciones Diversas que exijan seguridad....337
21.2.6.1 Aplicaciones concurrentes....338
21.2.6.2 Uso moderado de recursos de una supercomputadora....338
21.2.7 Aplicaciones que requieran cambiar la política de seguridad en tiempo de ejecución....338
21.2.7.1 Compra/venta de servicios automatizados en tiempo real....338
21.2.7.2 Actualización de permisos en sistemas que no se pueden interrumpir....339
CAPÍTULO XXII....341
CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS....341
22.1 CONCLUSIONES....341
22.2 RESULTADOS A DESTACAR DENTRO DE LOS DIFERENTES ASPECTOS DE LA ARQUITECTURA DE SEGURIDAD SAHARA....343
22.2.1 Del Diseño General de SAHARA....343
22.2.1.1 Diseño de una arquitectura genérica e integral para sistemas de agentes móviles....343
22.2.1.2 Uso de lugares para definir grupos afines que compartan recursos....343
22.2.1.3 Uso de regiones y grupos de regiones para crear redes seguras....343
22.2.1.4 Uso de ficheros Jar para transportar agentes más eficazmente....344
21.2.1.5 Uso de un mecanismo de validación de acceso al sistema....344
22.2.2 Protección del Servidor....344
22.2.2.1 Autenticación individual de autoridades propietarias de agentes....344
22.2.2.2 Especificación de un repositorio de Claves y Certificados encriptado....344
22.2.2.3 Autorización de privilegios individual mediante dominios de protección para cada autoridad o sistema remoto....345
22.2.2.4 Protección de recursos individuales con privilegios de acceso específicos mediante una política de seguridad versátil....345
22.2.2.5 Posibilidad de compartir recursos de forma protegida en tiempo de ejecución....345
22.2.2.6 Control del consumo de recursos por cada agente y por cada autoridad remota....345
22.2.2.7 Creación de dominios de protección para la asignación de privilegios....345
22.2.3 Transmisión segura....346
22.2.3.1 Autenticación de servidores....346
22.2.3.2 Transmisión de agentes protegida....346
22.2.4 Protección del Agente....346
22.2.4.1 Código del agente protegido mediante firmas digitales....346
22.2.4.2 Protección del estado de datos mediante diversas técnicas....346
22.2.5 Autocontrol del Sistema....347
22.2.5.1 Modificación de la política de seguridad en tiempo de ejecución....347
22.2.5.2 Uso de un mecanismo de control inteligente para mantener la estabilidad del sistema....347
22.2.6 Del Diseño e Implementación de MILENIO....347
22.2.6.1 Diseño modular, flexible y ampliable....347
22.2.6.2 Identificadores únicos en tiempo y espacio....347

22.2.6.3 Trasferencia selectiva y eficiente de clases 347
22.2.6.4 Aceleración del envío de agentes utilizando técnicas de compresión 348
22.2.6.5 Uso de itinerarios modificables en tiempo de ejecución para definir el recorrido de los agentes 348
22.2.6.6 Uso de una interfaz gráfica de alto nivel para la gestión y monitorización del sistema de agentes y para la creación de los mismos 348
22.2.7 Aplicación de los resultados de SAHARA a otros Sistemas de agentes 348
22.2.7.1 Mejora al modelo de seguridad de los aglets 348
22.2.7.2 Mejoras a la seguridad de los sistemas de agentes móviles en general 349
22.2.8 Evaluación de MILENIO 349
22.2.8.1 MILENIO es más versátil y cuenta con mayor número de capacidades de seguridad que los sistemas de agentes actuales 349
22.2.8.2 MILENIO cuenta con un desempeño aceptable al ofrecer una arquitectura de seguridad completa. 349
22.2.9 Aplicaciones de MILENIO 349
22.2.9.1 Posibilidad de aplicación flexible a un gran rango de aplicaciones 349
22.2.9.2 Ofrece nuevas oportunidades para el comercio electrónico. 350
22.2.9.3 Uso en aplicaciones que requieran cambiar su política de seguridad en tiempo de ejecución. 350
22.2.3 LINEAS DE INVESTIGACIÓN FUTURAS 350
22.2.3.1 Mecanismo de distribución de claves públicas entre servidores 350
22.2.3.2 Protección de datos del agente 350
22.2.3.3 Intercambio de mensajes e información entre agentes de forma segura 351
22.2.3.4 Interoperabilidad con otros sistemas de agentes móviles 351
FUENTES DE INFORMACION 352
BIBLIOGRAFIA: 352
FUENTES DE INFORMACION EN INTERNET: 359

Introducción

Hace unos pocos años apareció la tecnología de Agentes Móviles ofreciendo un nuevo paradigma para la programación de Internet, cada día se le concede mayor importancia y va teniendo más impacto dentro de la comunidad informática internacional. Hoy en día se augura que producirá una nueva generación de aplicaciones de software que revolucionarán la tecnología cliente/servidor usada en las aplicaciones de Internet y que producirá nuevos esquemas de programación del web que serán la tendencia dominante durante el próximo siglo.

El crecimiento exponencial de Internet así como la proliferación masiva de redes modificó los patrones básicos de programación monousuaria para generar sistemas de información distribuidos en donde se podía acceder a bases de datos distribuidas en distintos nodos de la red.

Más tarde se vería la conveniencia de tener el código cerca de los datos para acelerar el tiempo de respuestas en las consultas de información por lo que se originan diversas tecnologías de objetos distribuidos como CORBA y DCOM. Éstas tecnologías con todos sus beneficios sentaron las bases de las tecnologías de código móvil que representan una evolución de las mismas. No obstante, tanto las implementaciones del estándar CORBA y DCOM se enfrentaron a un mismo problema, el desempeño dentro de una intranet era aceptable, pero cuando la aplicación estaba distribuida en Internet, el desempeño de la aplicación disminuía en gran medida.

Dicho problema y la creciente demanda de aplicaciones de Internet cada vez más eficientes y sofisticadas dieron origen a lenguajes multiplataforma como Java que introdujo el concepto de applet que permitió la creación de aplicaciones que trasmitían el código para ser ejecutadas en nodos remotos de la red con lo que se lograba una interacción local y un mejor desempeño en el acceso a los datos. Sin embargo, los applets sólo podían viajar una vez desde el servidor de origen hacia el destino que lo solicitaba, además el cliente tenía que solicitar manualmente su envío pulsando en una página web.

Las demandas de autonomía y de eficiencia en la recopilación y proceso de la información de Internet propició la búsqueda de una nueva tecnología que permitiera optimizar los sistemas cliente/servidor y las tecnologías de objetos distribuidos cuyo principio fundamental consistía en: *llevar la computación a todos los destinos, en vez de traer todos los destinos a la computación.*

De esta forma apareció la tecnología de agentes móviles que permite la creación de agentes (objetos) que pueden moverse autónoma y libremente por distintos servidores de Internet para procesar tareas, recopilar y filtrar datos y finalmente regresar al sitio de origen con los resultados obtenidos. Permitiendo con ello, autonomía en el proceso de la información y mejor desempeño de las aplicaciones al permitir la interacción local en el proceso de la información en vez de mantener una conexión abierta a través de la red.

A pesar del gran avance que ha tenido en los últimos tiempos, la expansión de la tecnología se ha visto menguada por los problemas de portabilidad y de seguridad que la ejecución de código móvil de otros servidores implica. Los problemas relacionados con la portabilidad del código han sido medianamente solucionados con Java quedando sólo una gran barrera a vencer: **la seguridad**.

Esta tesis propone una arquitectura de seguridad integral para los sistemas de agentes móviles (SAHARA) que ofrezca un entorno de ejecución seguro en todo sistema que la implemente y que solucione en gran medida el problema de seguridad que se le ha imputado a la tecnología.

La originalidad del trabajo radica en crear diversas técnicas de protección contra todos los ataques que un sistema de agentes móviles puede sufrir, logrando con ello un trabajo único que ofrece una solución integral a los problemas de seguridad que tienen los sistemas de agentes móviles actuales.

La tesis describe detalladamente las medidas necesarias para proteger al servidor de agentes contra ataques de agentes móviles y de servidores remotos, especifica las medidas necesarias para proteger la transmisión entre servidores de agentes así como para proteger al agente móvil contra ataques de servidores maliciosos. Es importante resaltar sobre todo la técnica para proteger el estado de datos del agente móvil durante su itinerario ya que nadie ha propuesto una solución al problema que sea implementable como la que se ofrece en este trabajo.

Para demostrar la arquitectura propuesta en esta tesis se ha desarrollado un prototipo que implementa casi en su totalidad los medios de protección propuestos. El sistema se ha desarrollado utilizando la plataforma de Java2 v 1.2.1.

ORGANIZACIÓN DE LA TESIS

La primera parte de este trabajo realiza un análisis de las tecnologías de objetos distribuidos por ser éstas las predecesoras de los agentes móviles y porque ellas se encuentran sus orígenes.

La segunda parte introduce los conceptos de la tecnología de Agentes Móviles, explica los beneficios y ventajas de usar agentes móviles así como sus posibles aplicaciones. Después realiza un análisis y comparación de las diferentes plataformas para el desarrollo de agentes móviles, y ofrece un análisis concienzudo de sus carencias. En el siguiente capítulo se describe como se pueden incorporar los agentes móviles a los sistemas orientados a objetos tradicionales. Finalmente en su último capítulo analiza los

requerimientos que debe tener un sistema de agentes móviles, haciendo énfasis en la seguridad.

La tercera parte describe la arquitectura de seguridad integral propuesta en este trabajo. El capítulo XI ofrece una visión global de la arquitectura de seguridad SAHARA, del capítulo XII al XIV se describen todas la técnicas de protección usadas para ofrecer una seguridad integral. Finalmente en el capítulo XV se describen los alcances y las limitaciones de la arquitectura actual.

La cuarta parte describe la implementación del prototipo MILENIO. El capítulo XVI describe de forma general los componentes del sistema de agentes móviles. El capítulo XVII describe en detalle la implementación del núcleo del sistema de agentes móviles mientras que el XVIII describe en detalle la implementación e incorporación del sistema de seguridad al núcleo del sistema de agentes. Finalmente el capítulo IXX muestra el entorno integrado de desarrollo que ofrece el sistema de Agentes Móviles MILENIO.

La quinta y última parte explica los resultados de este trabajo. El capítulo XX describe el trabajo relacionado internacionalmente con esta investigación y la aplicación de la arquitectura de seguridad SAHARA a otros sistemas de agentes móviles en el mundo. El capítulo XXI realiza una evaluación del sistema de agentes móviles MILENIO, compara sus características de seguridad con otros sistemas así como su desempeño con el sistema de los aglets. Posteriormente explica algunos de los ámbitos de aplicación en los que puede ser usado el sistema.

Por último el capítulo XXII señala las conclusiones y resultados mas relevantes de este trabajo, y se describen la líneas de investigación futuras para ampliar y perfeccionar el modelo de seguridad.

Como último apartado de la tesis se incluyen varios apéndices. El apéndice A contiene el manual de usuario del sistema. El apéndice B contiene la API del sistema de agentes en donde se describen cada una de la clases que fueron diseñadas con sus parámetros respectivos. El apéndice C incluye algunos programas de ejemplo de agentes móviles para que los nuevos usuarios del sistema se puedan familiarizar rápidamente con el desarrollo de aplicaciones.

Por último se presente un glosario, la bibliografía un índice alfabético por palabras.

PARTE I.- PREDECESORES DE LOS AGENTES MÓVILES: OBJETOS DISTRIBUIDOS

Capítulo I

LA SIGUIENTE REVOLUCION CLIENTE/SERVIDOR

La siguiente generación de sistemas cliente/servidor será inevitablemente construida usando objetos distribuidos ya que empaquetados como componentes representan un ente con inteligencia propia que puede interoperar a través de sistemas operativos, redes, lenguajes y diversos vendedores de hardware. Solo esta tecnología ofrece los requerimientos necesarios para la interacción de software en un ámbito mundial.

1.1 REQUERIMIENTOS DE UN SOFTWARE CLIENTE/ SERVIDOR

Para que la tecnología cliente/servidor funcione apropiadamente es necesario que el medio que la soporte tenga:

- *Rico proceso de transacciones*: además de soportar una gran cantidad de transacciones, el nuevo ambiente requiere que las transacciones anidadas puedan expandirse a través de múltiples servidores y que se ejecuten en un largo periodo conforme se mueven de servidor en servidor.
- *Agentes móviles*: el nuevo ambiente requiere agentes electrónicos de todo tipo, quienes estarán colectando información para la administración del sistema o estadísticas.
- *Rico manejo de Datos*: esto incluye documentos compuestos que se puede mover, guardar, y editar en cualquier lugar de la red.
- *Entidades inteligentes que se gestionan a sí mismas*: se necesita software distribuido que sepa como gestionarse y configurarse a si mismo.
- *Software intermedio (de la lengua inglesa Middleware) inteligente*: el entorno distribuido debe proveer la semblanza de una imagen de un sistema individual a través de millones de maquinas híbridas cliente/servidor.

Hoy en día existen cuatro tecnologías que compiten para el desarrollo de nuevas aplicaciones cliente/servidor [ORFA96] como lo son: bases de datos SQL, monitores TP, Groupware y objetos distribuidos. Todas estas tecnologías ofrecen herramientas que facilitan la creación de aplicaciones cliente/servidor. Sin embargo como veremos muy brevemente algunas ofrecen más ventajas que otras.

1.2 CLIENTE/SERVIDOR CON BASES DE DATOS SQL:

Los servidores de bases de datos SQL son el modelo dominante para crear aplicaciones cliente/servidor. SQL comenzó como un lenguaje declarativo para la manipulación de datos usando comandos simples. Sin embargo las necesidades del mercado le demandaron más funciones para manipular datos. Para mejorar su deficiente desempeño encapsularon los comandos SQL en un procedimiento que reside en el mismo servidor que la bases de datos. A este mecanismo se le llama procedimiento almacenado.

Hoy en día todos los vendedores de SQL proveen algunas formas de procedimiento almacenado que son usadas para lograr la integridad de datos, la ejecución y mantenimiento del sistema así como la lógica de la aplicación del lado del servidor. Su mayor problema es que todas las extensiones no son estándar.

Lo mejor de SQL es que es muy fácil de usar y por consiguiente es sencillo implementar aplicaciones cliente/servidor bajo un entorno de un único vendedor usando sus herramientas visuales. Mas aún SQL es familiar a millones de programadores. Sin embargo SQL presenta grandes deficiencias como:

- *SQL es muy pobre en la gestión de procesos:* esto se debe a que la lógica de la aplicación esta en procedimiento almacenado, los programas se encuentran en segundo termino y los procesos hasta una tercera instancia.
- *El software intermedio de SQL no es estándar:* El SQL de diferentes vendedores no interopera entre ellos.
- *SQL no es apropiado para la gestión de datos complejos:* SQL está basado en tipos de datos simples por lo que no es apropiado para la gestión de datos complejos o datos que se encuentren distribuidos a través de varios servidores.

Resumiendo, es fácil de percibir que SQL es demasiado propietario y por consiguiente no apropiado para los fines que perseguimos.

1.3 CLIENTE/SERVIDOR CON MONITORES TP:

Los monitores TP fueron creados para gestionar procesos y para orquestar programas. Ellos lo logran al fraccionar aplicaciones complejas en piezas de código llamadas transacciones. Usando estas transacciones, un monitor TP puede obtener piezas de software que no saben nada unas de otras para actuar uniformemente.

Los monitores TP fueron introducidos para ejecutar aplicaciones que pueden servir a miles de clientes. Ellos lo logran proveyendo un entorno en donde ser intersectan a sí mismos entre los clientes y servidores, de esta manera ellos pueden gestionar transacciones, enrutarlas a través de sistemas, balancear su ejecución y restaurarlas después de fallas. Un monitor TP puede gestionar las transacciones en un servidor individual o a través de múltiples servidores y puede cooperar con otra federación de monitores TP.

Los monitores TP no tuvieron el éxito esperado debido a que los vendedores fueron muy lentos al dar a conocer la tecnología y los beneficios que ésta podría ofrecer.

Esta tecnología se transformará en objetos distribuidos, quizá la siguiente generación de monitores TP sea llamada ORB. Los vendedores de monitores TP están muy involucrados en la creación de estándares para el servicio de transacciones de CORBA.

Los tecnología de objetos puede aportar algo a los monitores TP, por ejemplo les facilita la creación y la gestión de modelos con abundantes transacciones, tales como transacciones anidadas y transacciones de larga duración.

1.3 CLIENTE/SERVIDOR CON GROUPWARE:

Groupware es una colección de tecnologías que nos permiten representar procesos complejos que se centran en actividades humanas. Esta construida sobre cinco tecnologías: gestión de documentos multimedia, flujo de información, correo electrónico, conferencia y programación. Groupware es un nuevo modelo cliente/servidor.

Groupware recopila estructuras de datos de alto nivel, que incluyen texto e imágenes y los organiza en algo llamado documento. Los documentos puedes ser duplicados, grabados o direccionados a cualquier lugar de la red. El documento multimedia es la unidad básica de gestión. Un buen ejemplo de esta tecnología es el *Lotus Notes.* De hecho Groupware ofrece una tecnología flexible que adapta a la forma de hacer negocios de la gente una forma de trabajo estructurada y que se puede configurar para satisfacer cualquier necesidad.

No obstante el software intermedio en el que Groupware esta construido es propietario y no es bien escalable para aplicaciones con muchas transacciones. Adicionalmente no trabaja bien con aplicaciones heredadas.

Debido a esto creemos que los objetos distribuidos incluirán a Groupware ya que los objetos proveen un software intermedio mas robusto para proveer funciones como las de *Lotus Notes*.

1.4 CLIENTE/SERVIDOR CON OBJETOS DISTRIBUIDOS

Con la infraestructura y empacado apropiado, los objetos ayudarán a subdividir las aplicaciones cliente/servidor en componentes que son capaces de gestionarse a sí mismos, que pueden interaccionar juntos y navegar a través de redes y sistemas operativos.

Los objetos distribuidos son apropiados para crear sistemas cliente/servidor porque los datos y la lógica de la aplicación son encapsulados dentro de los objetos, permitiéndoles estar localizados en cualquier parte del sistema distribuido. Los objetos distribuidos tienen el potencial inherente de permitir componentes granulares de software que interoperan entre redes y se ejecutan en diferentes plataformas, coexisten al heredarse a través de varias aplicaciones y gestionan por sí mismos los recursos que ellos controlan.

Sin embargo los objetos distribuidos no son todo, ellos requieren ser empacados como componentes que puedan interaccionar en conjuntos. Estas componentes combinarán lo mejor de las tecnología cliente/servidor y de objetos distribuidos. Ellas permitirán construir sistemas de información enteros al ensamblar componentes de objetos.

Después de hacer un análisis concienzudo es fácil apreciar que los objetos distribuidos es la plataforma que más ventajas ofrece para desarrollar este tipo de aplicaciones por lo que le dedicaremos algunos capítulos más de éste trabajo al análisis de esta tecnología. Veremos con detalle la tecnologías de CORBA, Opendoc y COM

Capítulo II

DE OBJETOS DISTRIBUIDOS A COMPONENTES INTELIGENTES

Un objeto distribuido es un ente con inteligencia que puede vivir en cualquier parte de una red. Los objetos distribuidos son empaquetados como piezas independientes de código que pueden ser accedidas por clientes remotos a través de invocaciones a sus métodos. Un componente es un objeto que no pertenece a un programa en particular, lenguaje o implementación pero que provee el mejor medio para las aplicaciones distribuidas. Se describirá con más detalle cada uno de ellos.

2.1 OBJETOS DISTRIBUIDOS

Los objetos distribuidos son piezas inteligentes de software que pueden intercambiar mensajes transparentemente desde cualquier parte del mundo [ORFA96]. El lenguaje y compilador que se usa para crear servidores de objetos distribuidos son totalmente transparentes a sus clientes, éstos no necesitan saber donde reside el objeto distribuido ni el sistema operativo en el que se ejecuta; y pueden estar en la misma maquina o bien en cualquier lugar de una red mundial.

Además de esto podemos añadir que los objetos distribuidos por ser objetos gozan de todas las bondades que éstos proveen, por lo que cualquier sistema de objetos proporciona las tres propiedades básicas : encapsulación, herencia y polimorfismo. Estas propiedades le permiten al objeto proporcionar sólo la interfaz deseada a sus clientes; reutilizar código y comportarse de distintas maneras.

Sin embargo el interés fundamental de los objetos distribuidos está en cómo esta tecnología puede ser extendida para enfrentarse con problemas complejos que son heredados al crear sistemas cliente/servidor robustos, es decir como los objetos pueden trabajar juntos a través de una máquina y los limites de una red para crear soluciones cliente/servidor. Nos encaminamos a los sistemas cliente servidor basados en objetos en donde se apreciara lo mejor de la grandeza y potencial de éstos.

2.2 COMPONENTES

Los componentes son objetos individuales que pueden integrarse por sí mismos en redes, aplicaciones, lenguajes y sistemas operativos. Los objetos distribuidos son, por definición componentes debido a la forma en como son empaquetados. En sistemas de objetos distribuidos la unidad de trabajo y distribución es el componente. Sin embargo nótese que no todos los componentes son objetos y no todos son distribuidos.

En resumen podemos decir que los componentes reducen la complejidad de una aplicación, el costo de desarrollo y tiempo para meter un producto al mercado. Ellos también mejoran la reusabilidad, mantenimiento e independencia de la plataforma del software, así como la distribución en cliente/servidor. Finalmente los componentes proveen más libertad de elección y flexibilidad.

Los componentes interoperan usando un lenguaje neutral en un modelo de interacción cliente/servidor. A diferencia de los objetos tradicionales los componentes interoperan a través de lenguajes, sistemas operativos y redes, pero también son ellos como los objetos en el sentido que soportan herencia, polimorfismo y encapsulación. Existen algunos componentes que no pueden ser extendidos a través de la herencia llamados cajas negras.

Debido a que los componentes pueden ser definidos en forma distinta por varias personas, se definirán en base a las funciones mínimas que deben proveer:

- *Es una entidad comercial:* es una pieza binaria de software autocontenida que típicamente se puede comprar en un mercado abierto.
- *No es una aplicación completa:* esta diseñado para desempeñar un conjunto limitado de tareas dentro de un dominio de una aplicación.
- *Puede ser usado en combinaciones impredecibles.*
- *Tiene una interfaz bien definida:* un componente sólo puede ser manipulado a través de su interfaz. Un componente CORBA/OpenDoc también provee un lenguaje de definición de interfaces (IDL) que se puede usar para invocar al componente, heredarlo o sobrescribir sus funciones.
- *Es un objeto interoperable:* puede ser invocado como un objeto en un espacio de direcciones, redes y sistemas operativos.
- *Es un objeto extendido:* ya que soportan encapsulación, herencia y polimorfismo.

En resumen un componente es una pieza de software autocontenida, que se puede reusar y que es independiente de cualquier aplicación.

2.3 SUPERCOMPONENTES

Los supercomponentes son componentes con inteligencia añadida. La inteligencia es requerida para crear objetos autónomos, débilmente acoplados y empaquetados que puedan navegar a través de las máquinas y vivir en las redes. Consecuentemente estos componentes necesitan proveer:

- *Seguridad.*
- *Licencia.*
- *Versión.*
- *Manejo de su ciclo de vida.*
- *Soporte para barras de herramientas abiertas:* un componente debe permitirse a si mismo el ser importado dentro de una barra de herramientas estándar.
- *Notificación de eventos:* un componente debe notificar a los grupos interesados que algo de interés le ha sucedido.
- *Configuración y manejo de propiedades:* debe proveer una interfaz que permita configurar sus propiedades y códigos.
- *Codificación:* debe permitir que su interfaz sea controlada vía lenguajes de codificación
- *Metadato e introspección:* al requerírsele debe proporcionar información sobre sí mismo, que incluye una descripción de sus interfaces, atributos y de las conexiones de software que soporta.
- *Control y bloqueo de transacciones.*
- *Persistencia.*
- *Relaciones:* un componente debe ser capaz de formar asociaciones dinámicas y permanentes con otros componentes.
- *Fácil de usar.*
- *Autoverificable:* debe probarse para ser capaz de determinar sus propios problemas.
- *Semántica de mensajes:* debe comprender el vocabulario del conjunto de componentes al que pertenece y extensiones del dominio especifico que soporta.
- *Debe instalarse a sí mismo.*

Esta lista proporciona una buena idea del nivel de calidad y funcionalidad que se esperan de los componentes. Una buena noticia es que OpenDoc/CORBA y OLE/COM ya proveen varias de estas características.

La última meta será crear componentes que se comporten como objetos de negocios; estos son componentes que modelan sus contrapartes del mundo real en algún dominio del nivel de aplicación. Ellos colaboran a que un trabajo se realice.

Capítulo III

CORBA (Common Object Request Broker Arquitecture) 2.0

El objetivo inicial y primordial del grupo de gestión de objetos (OMG de Object Management Group) es crear una verdadera infraestructura abierta para la distribución de objetos en vez de una aplicación comercial controlada por una compañía.

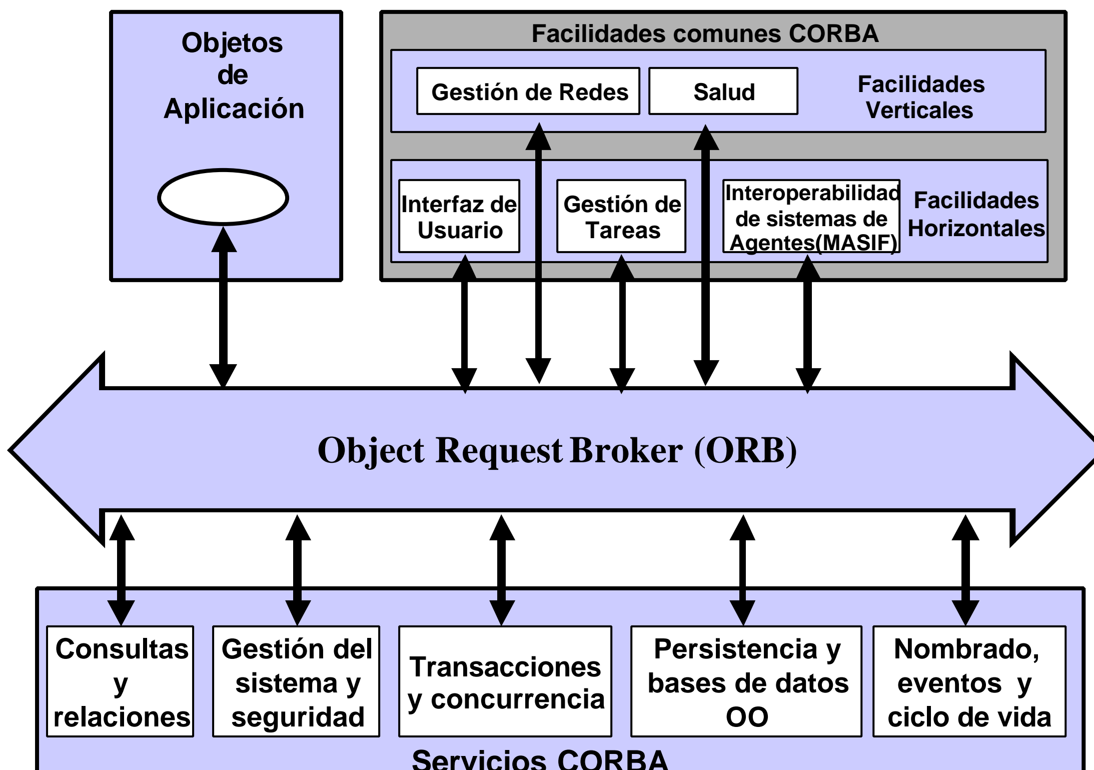


Figura 3.1 - La Arquitectura de CORBA

El secreto de su éxito ha sido que OMG ha creado las especificaciones de una interfaz, no el código. Las interfaces que especifica son siempre derivadas de una demostración

tecnológica llevada a cabo por las compañías miembro. Las interfaces son escritas en un lenguaje de definición de interfaces (IDL) que define los limites de los componentes.
La arquitectura común para la gestión de peticiones de objetos CORBA, consta de un bus de objetos, Servicios de objetos comunes, facilidades comunes y aplicaciones de objetos [ORFA96], como se pueden apreciar el la figura 3.1. Veamos pues como esta constituida esta arquitectura así como su funcionamiento.

3.1 EL BUS DE OBJETOS ORB(OBJECT REQUEST BROKER)

Un ORB CORBA 2.0 provee los mecanismos requeridos por los objetos para comunicarse con otros objetos a través de lenguajes, herramientas, plataformas y redes heterogéneas. También provee el entorno para gestionar estos objetos, advertir su presencia y describir sus metadatos. Un ORB CORBA es un bus de objetos que se describe así mismo.

Usando un ORB, un objeto cliente puede invocar transparentemente un método en un objeto servidor, el cual puede estar en la misma máquina o a través de la red, el ORB intercepta la llamada y es responsable de encontrar el objeto que implemente la petición, pasarle los parámetros, invocar al método, y regresar los resultados. Los papeles de cliente/servidor son sólo usados para controlar las interacciones entre los objetos, ya que los objetos en el ORB pueden actuar a la vez como cliente y como servidor.

3.1.1 La anatomía de un ORB CORBA 2.0

Dentro de un ORB interaccionan varios componentes para lograr la comunicación entre los objetos. (como se muestra en la figura 3.1.1). En principio señalaremos lo que hace CORBA en el lado del cliente :

- *Los stubs[1] IDL del lado del cliente :* proveen la interfaz estática de los servicios del cliente. Estos stubs precompilados definen como los clientes invocan los servicios correspondientes en los servidores. Estos servicios son definidos usando un IDL, y ambos son generados por un compilador IDL. Un cliente debe tener un stub IDL para cada interfaz en el servidor. Este stub incluye código para realizar la comunicación y el paso de parámetros.
- *La interfaz de invocación dinámica (DII) :* permite descubrir el método a ser invocado en tiempo de ejecución.
- *Las APIs del repositorio de interfaces :* permiten obtener y modificar la descripción de todas las interfaces componentes registradas, los métodos que soportan y los parámetros que requieren. Es un repositorio de metadatos dinámico para ORBs.
- *La interfaz ORB :* consiste en unas cuantas APIs de servicios locales que pueden ser de interés para una aplicación. Como por ejemplo el convertir la referencia de un objeto a una cadena.

[1] Los stubs son fragmentos de código que se generan tanto en el lado del cliente como del lado del servidor.

El soporte de invocaciones cliente/servidor dinámicas y estáticas así como el repositorio de interfaces le dan a CORBA una ventaja sobre otros entornos de distribución de objetos. Las invocaciones estáticas son más rápidas y fáciles de programar. Las dinámicas proveen una máxima flexibilidad pero son difíciles de programar, aunque son útiles para las herramientas que requieren descubrir los servicios en tiempos de ejecución.

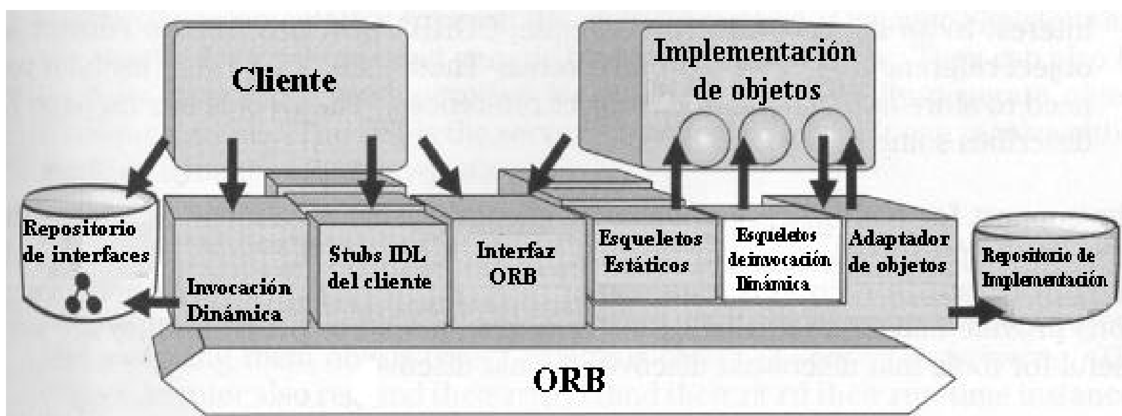


Figura 3.1.1 - Estructura de un ORB CORBA

El lado del servidor no puede identificar la diferencia entre una invocación estática o dinámica; en ambos casos, el ORB localiza un adaptador del objetos del servidor, le transmite los parámetros, y le transfiere el control a la implementación del objeto a través del stub IDL del lado del servidor[2]. A continuación se muestran los elementos de CORBA del lado del servidor.

- *Los stubs IDL del lado del servidor :* OMG los llama esqueletos, proveen la interfaz estática para cada servicio exportado por el servidor. Estos fragmentos, como los del lado del cliente son creados al usar un compilador IDL.
- *La interfaz de esqueletos dinámica :* proveen un mecanismo de unión en tiempo de ejecución para los servidores que necesitan gestionar las llamadas de métodos para componentes que no tiene un esqueleto base IDL compilado. Los esqueletos dinámicos son muy útiles para la implementación de puentes genéricos entre ORBs.
- *El adaptador de objetos :* provee el entorno en tiempo de ejecución para instanciar los objetos servidores, pasándoles peticiones y asignándoles un identificador de objeto (llamada referencia al objeto). El adaptador de objetos también registra las clases que soporta y sus instancias en tiempo de ejecución con el repositorio de implementación.
- *El repositorio de implementación :* provee un repositorio de información en tiempo de ejecución acerca de las clases que el servidor soporta, los objetos que están instanciados y sus identificadores. Guarda también información adicional relacionada con los ORBs.
- *La interfaz ORB :* consiste en unas cuantas APIs a servicios locales que son idénticas a los del lado del cliente.

[2] Llamado también esqueleto

3.1.2 Invocación de métodos CORBA. Estáticos contra Dinámicos

Existen dos tipos de invocaciones cliente/servidor que son soportadas por un ORB CORBA, estás son como se aprecia en la siguiente figura, las invocaciones estáticas y dinámicas

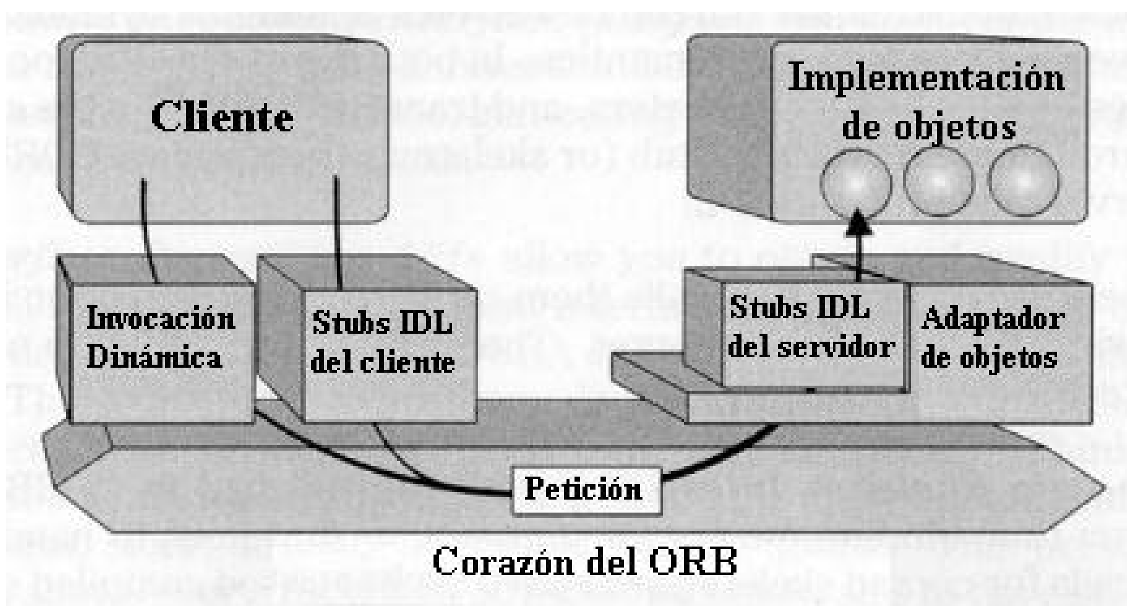


Figura 3.1.2 - Invocaciones estáticas y dinámicas de métodos

La interfaz estática es generada directamente en fragmentos por el precompilador IDL. Son ideales para programas que saben en tiempo de compilación las operaciones particulares que requerirán invocar. Tiene las siguientes ventajas sobre la invocación dinámica:

- Más fácil de programar.
- Provee un chequeo de tipos mas robusto.
- Buen desempeño.
- Es autodocumentada.

Los pasos que involucran un llamado estático son los siguientes :

1. *Definir las clases de objetos usando un lenguaje de definición de interfaces (IDL) :* Las IDLs son el medio por el cual los objetos le dicen a sus clientes potenciales que operaciones están disponibles y como deberían ser invocadas.
2. *Ejecutar el fichero IDL a través de un precompilador del lenguaje :* se procesa el fichero IDL y se producen los esqueletos para la implementación de clases servidoras.
3. *Añadir el código de la implementación a los esqueletos :* es decir crear las clases servidoras.

4. *Compilar el código :* se generan cuatro ficheros de salida que son : ficheros de importación que describen a los objetos en el repositorio de interfaces, stubs del cliente para los métodos en las IDLs definidas, stubs del servidor que llaman a los métodos en el servidor y el código que implementan las clases en el servidor.
5. *Unir las definiciones de clases a el repositorio de interfaces :* para que se pueda acceder a ellas en tiempo de ejecución.
6. *Instanciar los objetos en el servidor :* al arrancar, un adaptador de objetos de un servidor debe instanciar los objetos servidor que sirven para la invocación de métodos remotos del cliente. Éstos son instancias de las clases de aplicación del servidor.
7. *Registrar los objetos en tiempo de ejecución en el repositorio de implementación:* el adaptador de objetos registra en el repositorio de implementación la referencia a un objeto y el tipo de cualquier objeto que se instancia en el servidor. El ORB usa esta información para localizar los objetos activos o para requerir la activación de objetos en un servidor particular.

En contraste la invocación de métodos dinámica provee un entorno de mayor flexibilidad, permite añadir nuevas clases a el sistema sin requerir cambios en el código del cliente. Es muy útil para herramientas que descubren los servicios que son proveídos en tiempo de ejecución. Permiten escribir código genérico con APIs dinámicas.

Para invocar un método dinámico en un objeto, el cliente debe desempeñar los siguientes pasos :

1. *Obtener la descripción del método del repositorio de interfaces .*
2. *Crear la lista de argumentos :* la lista es creada usando la operación *create_list* y llamando *add_arg* tantas veces como se requiera para añadir cada argumento a la lista.
3. *Crear la petición :* la petición debe especificar la referencia al objeto, el nombre del método, y la lista de argumentos.
4. *Invocar la petición :* se puede invocar la petición de 3 maneras : 1) la llamada invocada envía la petición y obtiene los resultados ; 2) la llamada enviada devuelve el control al programa, el cual debe emitir un *get_response* ; 3) la llamada enviada puede ser definida de un sólo sentido.

Como se puede apreciar la invocación dinámica de un método requiere un cierto esfuerzo. Se cambia complejidad y desempeño por añadir flexibilidad.

3.1.3 El lado del servidor de CORBA. El adaptador de objetos

Lo que una implementación de objetos necesita en el lado del servidor es una infraestructura servidora que registre las clases de la implementación, instancias a nuevos objetos, que les de un único identificador, que advierta su existencia, que invoque sus métodos en las peticiones de los clientes y que gestione peticiones concurrentes a sus servicios. La respuesta a quién desempeña este tipo de trabajo es el adaptador de objetos.

El adaptador de objetos es el mecanismo primario para que una implementación de un objeto pueda acceder a los servicios de un ORB. Aquí están algunos de los servicios que provee.

1. Registrar las clases del servidor en el repositorio de implementación.
2. Instanciar los nuevos objetos en tiempo de ejecución.
3. Generar y gestionar la referencias únicas a los objetos.
4. Anunciar la presencia de servidores de objetos y sus servicios.
5. Manipular las llamadas provenientes de clientes.
6. Direccionar la llamada hacia el método apropiado.

Un adaptador de objetos define como son activados los objetos. Esto lo puede hacer creando un nuevo proceso, creando un nuevo enlace dentro de un proceso existente o usando un enlace o proceso existente. Un servidor puede soportar una variedad de adaptadores de objetos para satisfacer diferentes tipos de peticiones, CORBA especifica un **adaptador de objetos básico (BOA** de Basic Object Adapter**)** que puede ser usado por la mayoría de los objetos del ORB y que puede servir como un estándar.

CORBA requiere que un BOA este disponible en cada ORB y necesita que las siguientes funciones sean proveídas por la implementación del BOA.

- Un repositorio de implementación que permita instalar y registrar la implementación de los objetos.
- Mecanismos para generar e interpretar las referencias a objetos.
- Un mecanismo para identificar al cliente que hace la llamada.
- Activación y desactivación de objetos implementados.
- Invocación de métodos a través de fragmentos

Un BOA soporta aplicaciones tradicionales y orientadas a objetos. No especifica como son localizados y empaquetados los métodos. Para obtener la más amplia cobertura de aplicación, CORBA define cuatro políticas de activación de objetos que son:

- *Servidor BOA compartido:* múltiples objetos pueden residir en el mismo programa. El BOA activa el servidor la primera vez que una petición es realizada sobre cualquier objeto implementado por aquel servidor. El servidor gestiona una petición a la vez y avisa cuando ha terminado para procesar la siguiente.
- *Servidor BOA no compartido:* cada objeto reside en un proceso de servidor diferente. Un nuevo servidor es activado la primera vez que una petición es solicitada en el objeto. Un nuevo servidor es iniciado cada vez que se hace una petición a un objeto que aún no esta activado, aunque un servidor para otro objeto con la misma implementación este activo.
- *Servidor BOA por método:* un nuevo servidor es comenzado cada vez que una petición es hecha. El servidor se ejecuta solo por el tiempo que dura el método particular. Varios procesos de servidor para el mismo objeto pueden estar activos concurrentemente.
- *Servidor persistente BOA :* los servidores son activados por medios externos al BOA. Un vez iniciado el BOA trata a todas las llamadas subsecuentes como llamadas a servidores compartidos ; envía activaciones para objetos individuales y llamadas a métodos para un proceso individual.

3.1.4 Inicialización de CORBA 2.0 ó ¿Cómo un componente encuentra su ORB ?

Las llamadas que un objeto debe seguir para inicializarse así mismo son:

1. Obtener la referencia al objeto de su ORB .
2. Obtener el puntero a su adaptador de objetos.
3. Descubrir que servicios iniciales están disponibles.
4. Obtener las referencias a los objetos de los servicios que se desean.

Para cada uno de los pasos anteriores se debe llamar al método apropiado. Un objeto puede inicializarse así mismo en más de un ORB.

3.1.5 El ORB mundial y la arquitectura Inter-ORB

CORBA 2.0 añadió interoperabilidad al especificar un protocolo Inter-ORB de Internet (IIOP de Internet Inter-ORB Protocol). El IIOP es básicamente TCP/IP con algunos mensajes de intercambio definidos por CORBA que sirve como protocolo común en una linea principal (backbone). Cada ORB debe implementar un IIOP o proveer un puente hacia él.

A continuación se muestran los elementos de la arquitectura inter-ORB CORBA 2.0.

- *El protocolo Inter-ORB general (GIOP) :* especifica un conjunto de formatos para mensajes y representaciones de datos comunes para comunicaciones entre ORBs. El GIOP define siete formatos de mensajes que cubren toda la semántica de petición/respuesta del ORB.
- *El protocolo Internet Inter-ORB (IIOP) :* especifica como son los mensajes de GIOP intercambiados sobre redes TCP/IP. El IIOP hace posible el uso de internet como un ORB principal a través del cual otros ORBs pueden enlazarse.
- *Los protocolos de entorno especifico Inter-ORB (ESIOPs)* son usados para interoperar en redes especificas. CORBA 2.0 especifica a DCE como el primero de muchos ESIOPs opcionales. El DCE ESIOP incluye características avanzadas como seguridad, directorios particulares y globales, tiempo distribuido y autenticación RPC.

3.2 LOS METADATOS DE CORBA : IDL Y EL REPOSITORIO DE INTERFACES.

Los metadatos es el ingrediente que permite crear sistemas cliente/servidor ágiles. Un sistema ágil es autodescriptible, dinámico, y reconfigurable. El sistema ayuda a los

componentes a descubrirse unos a otros en tiempo de ejecución, y les provee información que le permite interoperar así como crear y gestionar componentes.
El lenguaje de metadatos de CORBA es el lenguaje de definición de interfaces (IDL) y el repositorio de metadatos de CORBA es el repositorio de interfaces que es una base de datos que se puede actualizar y cuestionar en tiempo de ejecución y que contiene la información generada por las IDLs.

3.2.1 Las IDLs de CORBA y su estructura

El IDL de CORBA es un lenguaje neutral y totalmente declarativo. Provee interfaces independientes del lenguaje de programación y sistema operativo para todos los servicios y componentes que residen en el bus CORBA.

Un contrato IDL incluye una descripción de cualquier servicio que un componente servidor quiera exponer a sus clientes. Los componentes servidores deben soportar dos tipos de clientes: 1) clientes que invoquen sus servicios en tiempo de ejecución ; y 2) desarrolladores que usan el IDL para extender las funciones de sus componentes existentes por medio de la herencia.

La gramática IDL es un subgrupo de C++ con palabras reservadas adicionales para soportar los conceptos de distribución, también soporta las características estándar de preprocesamiento de C++.

El repositorio de interfaces contiene metadatos que son idénticos a los componentes que se describen en la IDL. De hecho el repositorio de interfaces es simplemente un base de datos activa que contiene versiones compiladas de la información en metadatos, capturada vía IDL.

Los principales elementos para construir un IDL CORBA son:

- *Los Módulos* proveen un espacio de nombres (namespace) para agrupar a un conjunto de descripciones de clases, es decir agrupa varias clases.
- *Las Interfaces* definen un conjunto de métodos que los clientes pueden invocar en un objeto. Una interfaz puede tener atributos y puede declarar una o más excepciones .
- *Las operaciones* son el equivalente en CORBA de los métodos. Un parámetro tiene un modo que indica si un valor es pasado de el cliente al servidor (in), de el servidor al cliente (out), o en ambos(inout).
- *Los tipos de datos* son usados para describir los valores aceptados de los parámetros, atributos, y valores de retorno de CORBA. Estos tipos de datos son llamados objetos de CORBA que son usados a través de múltiples lenguajes, sistemas operativos y ORBs. El tipo *any* es muy útil en situaciones dinámicas ya que representa a cualquier tipo de datos IDL posible.

CORBA define códigos de tipos que representan cada tipo de datos de la IDL definida. Se usan para crear datos autodescriptibles que pueden ser pasados a través de ORBs y

repositorios de interfaces. Cada código de tipo tiene un ID de repositorio globalmente único. Son usados :

- Por interfaces de invocación dinámica.
- Por protocolos Inter-ORB.
- Por repositorios de interfaces.
- Por el tipo de datos any.

La interfaz *TypeCode* de CORBA define un conjunto de métodos que permiten operar con códigos de tipos, compararlos y obtener sus descripciones.

3.2.2 El repositorio de interfaces

Un repositorio de interfaces CORBA 2.0 es una base de datos en línea de definiciones de objetos. Esta definiciones pueden ser capturadas directamente de un compilador IDL o a través de las nuevas funciones de escritura del repositorio de interfaces. La especificación de CORBA detalla como la información es organizada y recuperada del repositorio. Lo hace creativamente al especificar un conjunto de clases cuyas instancias representan la información que está en el repositorio. La jerarquía de clases refleja la especificación IDL.

Un ORB utiliza las definiciones de objetos en el repositorio para :

- Proveer un chequeo de tipos de las definiciones de métodos.
- Ayuda a conectar ORBs.
- Provee información en metadatos para clientes y herramientas.
- Proporciona objetos autodescriptibles.

Los repositorios de interfaces pueden ser mantenidos localmente o gestionados como recursos departamentales o empresariales. Sirven como una fuente valiosa de información en la estructura de clases y en las interfaces de los componentes. Un ORB puede tener acceso a múltiples repositorios de interfaces.

El repositorio de interfaces es implementado como un conjunto de objetos que representan la información en él. Estos objetos deben ser persistentes. CORBA agrupa los metadatos en módulos que representan el espacio de nombres. Los nombres de los objetos del repositorio son únicos dentro de un módulo. CORBA define una interfaz para cada uno de sus ocho estructuras IDL :

- ModuleDef.
- InterfaceDef.
- OperationDef.
- ParameterDef.
- AttributeDef.
- ConstantDef.
- ExceptionDef.
- TypeDef.

En adición a estas ocho interfaces, CORBA especifica una interfaz llamada *Repository* que sirve como raíz para todos los módulos contenidos en un espacio de nombres del repositorio. Cada repositorio de interfaces es representado por un objeto *repository* raíz global.

Además de estas jerarquías contenidas CORBA definió tres superclases abstractas llamadas *IRObject, Contained,* y *Container*. Todos los objetos del repositorio de interfaces heredan de la interfaz *IRObject*, la cual provee un método *destroy*. Los objetos que son contenedores heredan operaciones de navegación de la interfaz *Container*. La interfaz *Contained* define el comportamiento de los objetos que son contenidos en otros objetos. Todas las clases del repositorio son derivadas de *Container,* de *Contained* o de ambas a través de herencia múltiple. Este astuto esquema permite al repositorio de objetos comportarse de acuerdo a sus relaciones de contención.

Para acceder a los metadatos del repositorio de interfaces se invocan métodos polimorficamente en diferentes tipos de objetos. Es posible navegar y extraer información del los objetos del repositorio con tan sólo nueve métodos. Cinco de estos métodos son derivados de las clases antecesoras *Container* y *Contained*. Los otros cuatro métodos son específicos a las interfaces *InterfaceDef* y *Repository*.

A partir de CORBA 2.0 se pueden crear federaciones mundiales de repositorios de interfaces que operen a través de ORBs múltiples. Para prevenir colisiones de nombres, estos repositorios asignan identificadores únicos a interfaces y operaciones globales. Para lograr esto se siguen las siguientes convenciones de nombrado :

- *Nombres con alcance* los cuales son hechos de uno o más identificadores separados por los caracteres ' : :'.
- *Identificadores de repositorio* que identifican globalmente módulos e interfaces. Son usados para sincronizar las definiciones a través de ORBs y repositorios, existen dos formatos para especificarlos : 1) usando nombres IDL con prefijos únicos y 2) usando identificadores únicos universales DCE.

3.3 LOS SERVICIOS DE CORBA : NOMBRADO, EVENTOS Y CICLO DE VIDA

El ORB provee los mecanismos básicos para gestionar las peticiones de objetos. Todos los otros servicios son proveídos por objetos con interfaces IDL que residen en la cima del ORB. Los cuatro servicios más básicos que se requieren para operar dentro de un entorno ORB son : Nombrado, negociación (trader), ciclo de vida y eventos. Estos servicios los proporciona el propio ORB (Véase figura 3.3.1). Cuando un objeto es iniciado, requiere de estos servicios básicos para encontrar otros objetos en el ORB.

3.3.1 El servicio de nombrado de objetos de CORBA

El servicio de nombrado de objetos es el principal mecanismo para que los objetos en un ORB localicen otros objetos. El servicio de nombrado convierte los nombres humanos en referencias a objetos. Se puede opcionalmente asociar uno o mas nombres con la referencia de un objeto.

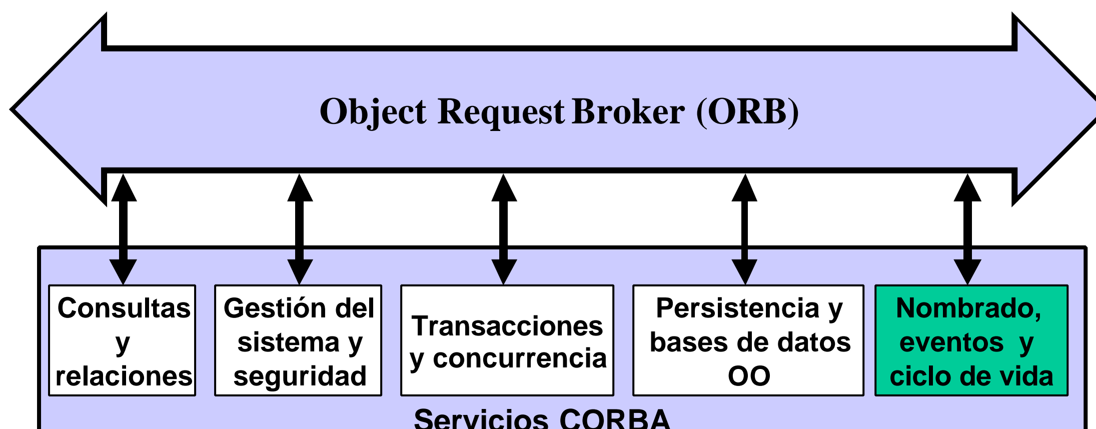


Figura 3.3.1 - Servicios de CORBA : Nombrado, eventos y ciclo de vida

Se puede referenciar un objeto CORBA usando una secuencia de nombres que forme un árbol de nombre jerárquico. Resolver un nombre significa encontrar el objeto asociado con el nombre en un contexto dado. Enlazar un nombre es crear una asociación nombre/objeto para un contexto particular.

Cada componente nombrado es una estructura con dos atributos : 1) un identificador que es el nombre del objeto ; 2) y un tipo que es una cadena en donde se puede poner una descripción del nombre del atributo.

Las interfaces *NamingContext* y *BindingInterator* implementan el servicio de nombrado. Se invoca el método *bind* en la interfaz *NamingContext* para asociar el nombre de un objeto con un contexto enlazado. De esta forma es como se implementa una jerarquía de nombrado. El método *list* permite iterar a través de un conjunto de nombres (regresa un objeto *BindingIterator*). Se puede iterar invocando *next_one* y *next_n* en el objeto regresado *BindingIterator*.

En resumen, el servicio de nombrado de objetos define las interfaces que permiten gestionar el espacio de nombres para los objetos, consultar y navegar a través de ellos. Los objetos que proveen estas interfaces viven en el ORB.

3.3.2 El servicio negociación (*trader*) de CORBA

En las próximas versiones de CORBA, el servicio de negociación de objetos proveerá un servicio que permita a un objeto localizar el servicio que más satisfaga sus necesidades. De acuerdo al RFP5, el nuevo proveedor de un servicio primero registrará

su servicio en el servicio de negociación, entonces dará toda la información relevante, incluyendo:

- Una referencia a un objeto.
- Una descripción del tipo de servicio que ofrece.

El objeto servicio de negociación almacena esta información persistentemente. Los clientes se ponen en contacto con el servicio de negociación para averiguar los servicios listados y preguntar por un servicio por su tipo. El servicio de negociación encontrará el más apropiado basado en el contexto del servicio requerido y la oferta de sus proveedores. Los servicios de negociación de diferentes dominios pueden crear federaciones.

3.3.3 El servicio de ciclo de vida de objetos de CORBA

El servicio de ciclo de vida de objetos provee operaciones para crear, copiar, mover y borrar objetos. Todas estas operaciones pueden ahora soportar asociaciones entre grupos de objetos relacionados. Esto incluye relaciones de referencia y contenido. El servicio de ciclo de vida provee interfaces que son derivadas del servicio de relaciones.

Para crear un nuevo objeto, un cliente en principio debe encontrar una fabrica de objetos, ejecutar una petición de creación, y recibir de regreso una referencia a un objeto. Lo puede crear también haciendo una copia de uno ya existente. Cuando un objeto es copiado a través de varias máquinas, la fabrica de objetos en el nodo destino es involucrado.

La interfaz de *LifeCycleObject* define las operaciones de copiar, mover, y eliminar. La interfaz de *FactoryFinder* define un método para encontrar fabricas de objetos y la de *GenericFactory* simplemente define una operación de creado de objetos.

Las interfaces compuestas de ciclo de vida añaden una nueva interfaz llamada *OperationsFactory*, la cual crea objetos de la clase *Operations.* Ésta soporta los métodos copiar, mover y borrar y añade un método para destruir.

Las tres clases *Node*, *Role*, y *Relationship* son derivadas de las clases correspondientes en el servicio de relación . Éstas permiten definir relaciones entre objetos que no saben nada uno de otro. Para definir una relación, primero se debe asociar un papel con un objeto y entonces crear las relaciones entre estos papeles.

3.3.4 El servicio de eventos de CORBA

El servicio de eventos permite a los objetos registrar y eliminar dinámicamente sus eventos específicos de interés. Un evento es una ocurrencia dentro de un objeto específico que puede ser de interés para uno o más objetos. Una notificación es un mensaje que un objeto envía a los grupos de interés para informarles que un evento

específico ha ocurrido sin conocerles, ya que esto normalmente es gestionado por el servicio de eventos.

El servicio de eventos mantiene la comunicación entre objetos. El servicio define dos papeles para los objetos: el de proveedores que son quienes producen los eventos y el de consumidores que los procesan a través de gestores de eventos. Los eventos se comunican con llamadas estándar CORBA. En adición, hay dos modelos para la comunicación de eventos de datos: empujar(*push*) y tirar(*pull*). En el modelo empujar el proveedor de eventos toma la iniciativa e inicia la transferencia de datos a los consumidores. En el modelo tirar, el consumidor toma la iniciativa y requiere los datos de eventos de un proveedor.

Un canal de eventos es un objeto estándar CORBA que se coloca en el ORB y mantiene la comunicación entre los proveedores y consumidores. El canal de eventos de objetos genérico no comprende el contenido de los datos que pasan. En vez de esto, los productores y consumidores están de acuerdo en una semántica común de eventos.

La interfaz de eventos soporta múltiples niveles de servicios. Un almacenamiento persistente de eventos debe ser proveído por el canal persistente de objetos como parte de sus servicios.

El servicio de eventos es extremadamente útil; ya que provee las bases para la creación de un nuevo género de aplicaciones de objetos distribuidos.

3.4 LOS SERVICIOS DE CORBA: TRANSACCIONES Y CONCURRENCIA:

OMG ha adoptado recientemente el servicio de transacciones de objetos (OTS de *Object Transaction Service*) el cual define interfaces IDL que permiten a múltiples objetos distribuidos en múltiples ORBs participar en transacciones atómicas (aún en presencia de una falla catastrófica). OTS opcionalmente soporta transacciones anidadas.

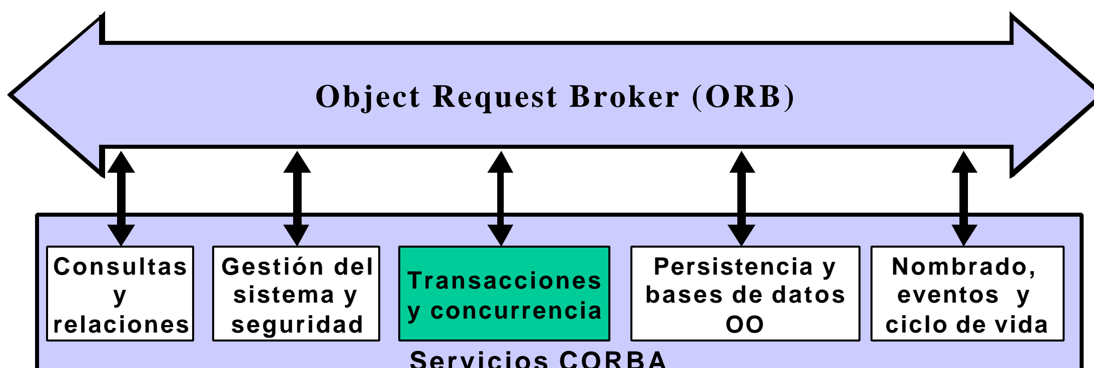


Figura 3.4.1 - Servicios de CORBA : Transacciones y concurrencia

El servicio de control de concurrencia permite a los objetos coordinar su acceso a recursos compartidos usando bloqueos. Cada bloqueo esta asociado con una fuente

individual y un cliente simple. El servicio define varios modos de bloqueo que corresponden a diferentes categorías de acceso a niveles múltiples de granularidad. Para apreciar la ubicación de los servicios dentro de CORBA observe la figura 3.4.1.

3.4.1 El servicio de transacción de objetos de CORBA

El servicio de transacción de objetos es posiblemente la pieza más importante del software intermedio (*middleware*) para objetos mundiales. Con OTS, los ORBs proveen un entorno favorable para correr componentes de misión critica. Sólo esta característica les da a los ORBs una ventaja sobre cualquier software intermedio cliente/servidor en competición.

Las transacciones han llegado a ser una filosofía de diseño de aplicaciones que garantizan robustez en sistemas distribuidos. La transacción es el contrato que une al cliente con uno o más servidores. Una transacción llega a ser la unidad fundamental de recuperación, consistencia, y concurrencia en un sistema de objetos distribuidos.

Los modelos de transacciones definen cuando comienza una transacción, cuando termina y cual es la unidad apropiada de recuperación en caso de una falla. Los nuevos modelos transaccionales (por ejemplo transacciones anidadas) proveen una granularidad mucho más fina de control sobre los diferentes enlaces que constituyen una transacción.

El OTS de CORBA provee las siguientes características:

- Soporte de transacciones simples y anidadas.
- Permite participar aplicaciones ORB y no-ORB en la misma transacción.
- Soporta transacciones que se expanden a través de ORBs heterogéneos.
- Soporta interfaces IDL existentes.

Los objetos involucrados en una transacción puede asumir uno de los tres papeles siguientes:

1. *Un cliente de transacciones* que ejecuta un conjunto de invocaciones de métodos que son marcadas con un principio y fin de transacción. El ORB intercepta el llamado inicial y lo dirige hacia el servicio de transacciones, el cual establece un contexto de transacciones asociado con el enlace del cliente. El cliente entonces ejecuta la invocación de un método usando objetos remotos. El ORB explícitamente sigue de cerca el contexto de transacciones y lo propaga en todas las comunicaciones subsecuentes entre los participantes en la transacción.
2. *Un servidor de transacciones* es la colección de uno o más objetos cuyo comportamiento es afectado por la transacción pero que no tiene estados o recursos de recuperación por sí mismo.
3. *Un servidor recuperable* es una colección de uno o más objetos cuyos datos son afectados por la ocurrencia de una transacción. Los objetos recuperables son objetos de transacciones con recursos de protección.

El OTS está directamente integrada con los mecanismos ORB. Él confía en que el ORB automáticamente propagará el contexto de la transacción. El contexto de la transacción es un pseudo objeto que es mantenido por el ORB y por cada enlace de software ORB.

Las cuatro interfaces principales del OTS son:

1. *Current* define un pseudo objeto de CORBA que les hace más fácil a los clientes el uso de OTS.
2. *Coordinator* es implementado por el servicio de transacciones. Los objetos recuperables lo usan para coordinar su participación en una transacción con el OTS.
3. *Resource* es implementado por el servicio de objetos recuperable para participar en una aceptación protocolaria de dos fases. El OTS usa este protocolo de dos fases para coordinar una aceptación de transacciones o abortarla a través de múltiples objetos servidor de tal forma que todos ellos fallen o tengan éxito.
4. *SubtransactionAwareResource* es implementada por un objeto servidor recuperable con un comportamiento de transacciones anidadas. Esta interfaz deriva de *resource.*

Para hacer un objeto de tipo transacción simplemente se hereda de la clase *TansactionalObject.*

3.4.2 El servicio de control de concurrencia de CORBA

El servicio de control de concurrencia provee las interfaces para fijar y liberar bloques que permitan a clientes múltiples coordinar su acceso a recursos compartidos. El servicio no define un recurso. Este servicio soporta modos de operación con transacciones y sin transacciones.

Un cliente del servicio de control de concurrencia puede escoger la forma de adquirir los bloqueos en una de dos formas :

- *A favor de una transacción*. En este caso, el servicio de transacciones se encarga de liberar los bloqueos cuando la transacción termina o aborta.
- *A favor de un cliente sin transacciones.* La responsabilidad de liberar los bloqueos recae sobre el cliente.

El servicio de control de concurrencia define un *lockset* que es un conjunto de bloqueos asociados con un recurso simple. A su vez define un coordinador de bloqueos que gestiona la liberación de los *locksets* relacionados.

Cuando una transacción anidada requiere un bloqueo que es mantenido por su padre, ésta se convierte en el nuevo propietario del bloqueo. Cuando una transacción anidada se ejecuta, el servicio de control de concurrencia automáticamente transferirá el propietario de todos sus bloqueos al padre de su transacción. Las interfaces para gestionar el control de concurrencia son :

- *LocksetFactory* permite crear *locksets.*
- *Lockset* permite adquirir y liberar bloqueos.
- *TransactionalLockset* soporta los mismos métodos que *Lockset*. La diferencia es que se debe pasar un parámetro que identifica a la transacción.

- *LockCoordinator* OTS invoca su método para liberar todos los bloqueos mantenidos por una transacción cuando aborta o se ejecuta.

3.5 LOS SERVICIOS DE CORBA : PERSISTENCIA Y BASES DE DATOS DE OBJETOS

La mayoría de los objetos distribuidos son persistentes. Esto significa que ellos deben mantener su estado tiempo después que el programa que los creo termine. Para lograrlo el estado del objeto debe estar almacenado en un almacenamiento de datos no volátil y buscar una forma de convertir datos de nuestro almacenamiento a objetos y viceversa.

Un sistema gestor de bases de datos orientadas a objetos ODBMS (de la lengua inglesa *Object DataBase Management System*) mágicamente mueve los objetos de memoria a disco para conocer las necesidades de uso, solo se ve un nivel simple de almacenamiento de objetos.

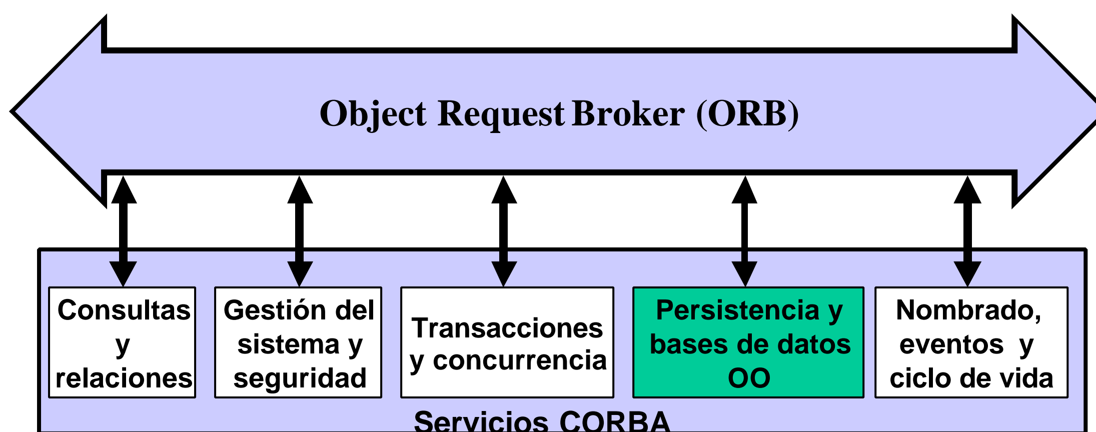


Figura 3.5.1 - Servicios de CORBA : Persistencia y bases de datos OO

3.5.1 El servicio de persistencia de objetos de CORBA (POS)

El servicio de persistencia de objetos POS (de *persistent Object Service*) permite a los objetos persistir mas allá de la aplicación que creo el objeto o el cliente que lo usa. POS permite al estado de un objeto ser salvado en un almacenamiento persistente y restaurarlo cuando sea necesitado.

El POS es el resultado de una mezcla que resultó en una especificación que puede proporcionar una variedad de servicios de almacenamiento incluyendo bases de datos SQL y ODBMSs. El POS define la interfaz a datos y a objetos persistentes usando interfaces definidas IDL. La idea principal detrás del POS es una interfaz de objetos simple a múltiples almacenamientos de datos.

El POS puede soportar almacenamientos de un nivel como ODBMSs y de dos niveles como bases de datos SQL. En un almacenamiento de un nivel, el cliente no esta al tanto de si un objeto esta en memoria o en disco. En contraste uno de dos niveles separa la memoria del almacenamiento persistente. El objeto debe ser explícitamente colocado de una base de datos a memoria y viceversa.

Algunas veces los clientes de un objeto necesitan controlar la gestión de la persistencia. En un extremo el servicio puede ser hecho transparente a las aplicaciones del cliente; en el otro las aplicaciones de los clientes pueden usar protocolos específicos de almacenamiento que muestran todos los detalles del mecanismo de almacenamiento persistente que lo soporta.

El objeto persistente (PO) esta a cargo de su persistencia ; él decide que protocolo de almacenamiento de datos usar y cuanta visibilidad le da a sus clientes. El PO puede delegar la gestión de su persistencia al servicio de persistencia o bien mantener todas sus iteraciones con el sistema de almacenamiento. El PO puede también heredar la mayoría de la funcionalidad que necesita para ser persistente.

Los elementos de un POS son:

- *Objetos persistentes(POs):* son objetos cuyo estado está almacenado persistentemente.
- *Gestor de objetos persistentes(POM):* es una interfaz de implementación independiente para las operaciones de persistencia.
- *Servicios de datos persistentes(PDSs):* son interfaces a las implementaciones de almacenamiento de datos particulares. Éstos mueven los datos entre los objetos y su almacenamiento.
- *Almacenamientos de datos:* son las implementaciones que almacenan los datos persistentes de un objeto independientemente de su espacio de direcciones en donde este contenido.

POS define tres protocolos de implementación específica para mover datos entre un objeto y diferentes PDSs :

- *El acceso directo(DA)* provee un acceso directo usando una IDL como un lenguaje de definición de datos.
- *El ODMG-93* provee acceso directo de C++ usando un lenguaje de definición de datos específico de ODMG.
- *El objeto de datos dinámico(DDO)* define una estructura que posee todos los datos para un objeto. Se puede usar para describir un objeto sin necesidad de una IDL y para que los clientes puedan formular consultas SQL dinámicas.

Las interfaces del servicio de persistencia son:

- *PIDFactory* permite crear un identificador de objeto persistente (PID) de tres maneras.
- *POFactory* permite crear una instancia de un objeto persistente.
- *PID* regresa una versión en tipo cadena de un PID.

- *PO* provee cinco operaciones que le permiten a un cliente controlar la relación de persistencia de un objeto con sus datos persistentes.
- *POM* provee cinco operaciones que un objeto persistente usa para comunicarse con su almacenamiento de datos.
- *PDS* provee las cinco operaciones que un POM usa para comunicarse con su almacenamiento de datos.

Además de estas interfaces existen las interfaces CLI del POS que encapsulan diversas funciones que permiten acceder a bases de datos relacionales, sistemas de ficheros Xbase y aún a bases de datos jerárquicas. Éstas interfaces permiten a los objetos acceder a bases de datos vía ORBs usando interfaces IDL definidas.

3.5.2 Sistemas gestores de bases de datos orientados a objetos (ODBMSs)

Los ODBMSs han tomado un cambio radical para compartir datos que están totalmente centrados en la gestión de objetos persistentes. Un ODBMS provee un almacenamiento persistente para los objetos en un entorno multiusario cliente/servidor. El ODBMS gestiona concurrentemente el acceso a los objetos, provee de bloqueos y protección para transacciones y cuida las tareas tradicionales como los respaldos. La diferencia con los sistemas relacionales es que los ODBMS almacenan objetos en vez de tablas. Utilizan PIDs que identificar de forma única a los objetos y los usan para crear relaciones referenciales y de contención.

Los ODBMSs usan la definición de clases y los lenguajes tradicionales de orientación a objetos para construir y definir el acceso a los datos e integran las capacidades de las bases de datos directamente en el lenguaje.

Por supuesto no todo es transparente al lenguaje, los ODBMSs requieren de un precompilador que procese las definiciones de objetos, extensiones del lenguaje y consultas. La salida del compilador debe ser enlazada a el ODBMS en tiempo de ejecución.

Las características que proporcionan los ODBMSs son las siguientes :

- Libertad para crear nuevos tipos de información.
- Rápido acceso a datos complejos.
- Vistas flexibles de estructuras compuestas.
- Estrecha integración con los lenguajes de orientación a objetos.
- Personalización de estructuras de información usando herencia múltiple.
- Soporte para transacciones con versión, anidadas y de larga duración.
- Repositorios para objetos distribuidos.
- Soporte para la gestión del ciclo de vida y objetos compuestos.

En general los ODBMSs tienen la ventajas sobre las bases de datos relacionales de conocer toda la estructura de un objeto complejo y en ocasiones hasta su comportamiento, y pueden referirlo por su identificador.

Existen varias arquitecturas alternativas para implementar ODBMSs en redes cliente/servidor, dos aproximaciones básicas son:

- *El servidor objeto* balancea la carga entre el cliente y el servidor. El cliente y el servidor se comunican moviendo los objetos del almacenamiento persistente del servidor a la memoria del cliente. Los clientes requieren objetos especificando su identificador de persistencia. Los métodos pueden ser ejecutados en el cliente o en el servidor, dependiendo de la localización del objeto. El proceso de consultas es hecho en el servidor. Esta es considerada la mejor implementación.
- *El servidor de paginas* es una aproximación de un gran cliente que trata los ODBMS como un almacenamiento multiusario de memoria virtual compartido. El servidor trae las paginas del disco y las mueve a la memoria cache del cliente. Los métodos y las consultas son ejecutados en el cliente en donde reside la lógica del objeto.

ODMG-93 es una extensión de el servicio de persistencia de objetos de CORBA que define como implementar un protocolo que provee un eficiente servicio de datos persistente(PDS) para objetos bien definidos. El estándar usa el modelo de objetos de OMG como su base y consiste de tres grandes componentes:

- Lenguaje de definición de objetos(ODL).
- Lenguaje de consultas de objetos(OQL).
- Enlaces a lenguajes como C++ y Smalltalk.

La única desventaja que se observa en los ODBMSs es la carencia de funcionalidad en las áreas de búsquedas complejas, optimizadores de consultas y escalabilidad de servidores. Más aún, algunos trabajan en el mismo espacio de direcciones que los programas de usuarios lo que implica que no hay protección entre la aplicación del cliente y el ODBMS.

3.6 LOS SERVICIOS DE CORBA: CONSULTA Y RELACIONES

Estos servicios permiten descubrir, crear, y manipular relaciones entre objetos. De igual forma permiten manipular múltiples objetos como un grupo. Al igual que los últimos dos grupos no forman parte de los servicios fundamentales, y su ubicación se puede apreciar en la siguiente figura.

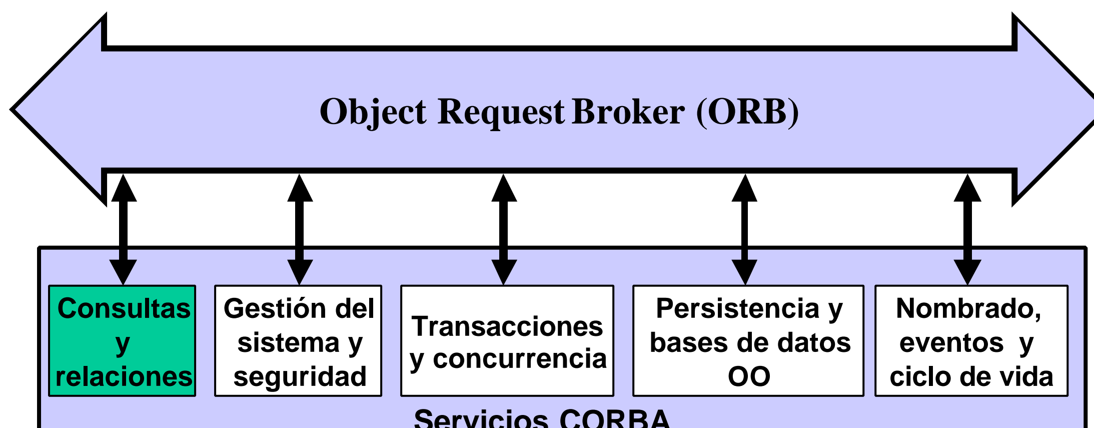


Figura 3.6.1 - Servicios de CORBA : Consultas y relaciones

3.6.1 El servicio de consultas de CORBA

En esencia este servicio permite encontrar objetos cuyos atributos coincidan con el criterio de búsqueda que se especifico en una consulta. Debe notarse que las consultas no tienen acceso al estado interno del objeto.

El servicio de consultas puede directamente ejecutar una consulta o delegarla algún otro evaluador de consultas. Éste servicio combina los resultados de la consulta de todos los evaluadores de consultas participantes y regresa los resultados finales a quien hizo la llamada.

Una consulta CORBA hace más que encontrar objetos ; permite también manipular una colección de ellos. El servicio de consulta trata a la colección misma como si fuera un objeto y define operaciones que permiten manipular y navegar en una colección. De igual forma permite añadir y eliminar miembros de la colección.

El servicio de consultas provee tres interfaces que permiten manipular los resultados de una consulta, estas son:

- *CollectionFactoy* crea una instancia de una colección vacía.
- *Collection* define operaciones que permiten añadir, remplazar y eliminar miembros de una colección.
- *Iterator* define tres operaciones que permiten invertir una colección.

Estas tres interfaces podrían ser las clases base para el servicio de colección de CORBA.

El servicio de consultas de CORBA provee cinco interfaces que trabajan en la preparación y ejecución de una consulta.

- *QueryEvaluator* evalúa una consulta.
- *QueryManager* es una versión más poderosa de la anterior que permite crear un objeto de consulta.

- *Query* define cuatro operaciones que se pueden ejecutar en una instancia de una consulta.
- *QueryableCollection* hereda su funcionalidad de las interfaces *QueryEvaluator* y *Collection*.

Es posible extender estas cuatro interfaces (vía herencia) para proveer funciones adicionales.

3.6.2 El servicio de colección de CORBA

Las colecciones permiten manipular a los objetos como un grupo. Normalmente se aplican operaciones en grupos en vez de aplicarlo a los objetos individuales que contengan. Algunos ejemplos podrían ser colas, pilas, listas, etc. Cada uno de estos tipos de colección exhiben comportamientos que son específicos a la colección.

El propósito del servicio de colección de CORBA es proporcionar una forma uniforme de crear y manipular las colecciones más comunes de tal forma que al proporcionar ciertas interfaces base se puedan agregar más funciones a través de la herencia.

3.6.3 El servicio de relación de CORBA

El servicio de relación de CORBA permite a los componentes y objetos que no saben nada uno de otro estar relacionados. Éste permite hacer esto sin cambiar los objetos existentes o requerir que añadan nuevas interfaces, en otras palabras el servicio permite crear relaciones dinámicas entre objetos inmutables. Sin el servicio de relación, los objetos tendían que mantener una lista de las relaciones usando punteros específicos a cada objeto.

Una relación es definida como un conjunto de papeles que dos o más objetos juegan. Un objeto puede jugar diferentes papeles en diferentes relaciones. El grado se refiere al número de papeles requerido en una relación. La cardinalidad define el número máximo de relaciones en el cual un papel está involucrado. Una relación puede tener uno o más atributos.

Una relación pueden también soportar invocación de métodos que regresen información. Observe que los atributos y métodos de una relación son totalmente independientes de los objetos que representa. Un conjunto de objetos relacionados forma una gráfica. Los objetos son nodos en la gráfica y las relaciones son las líneas entre ellos.

El servicio de relación define interfaces genéricas que permite asociar papeles y relaciones con objetos existentes CORBA, es posible invertirles. El servicio define dos relaciones específicas: contención y referencia. El servicio permite crear relaciones de grado y cardinalidad arbitraria y trata a las relaciones, papeles y gráficas como objetos

CORBA de primera clase por lo que se puede extender su funcionamiento generando subclases.

Lo mejor de todo es que es posible crear relaciones entre objetos que están inadvertidos de sus relaciones. Las interfaces del servicio de relación están agrupadas en tres categorías:

Las interfaces de relaciones base definen operaciones que permiten crear objetos con un papel y relación y navegar a través de las relaciones en las cuales participa un papel, en general son las siguientes:

- *IdentifiableObject* retorna verdadero si dos objetos CORBA son idénticos.
- *RelationshipFactory* permite crear una instancia de una relación.
- *RoleFactory* permite asociar un papel con un objeto CORBA.
- *Relationship* permite destruir una relación.
- *Role* permite navegar en las relaciones en las que participa un papel y enlazar un papel a una relación.
- *RelationshipIterator* permite iterar a través de relaciones adicionales en las que participa un papel.

Las interfaces de gráfica permiten describir e invertir gráficas, se mencionan a continuación:

- *Node* asocia un objeto con su papel.
- *Transversal* permite navegar a través de una gráfica de objetos relacionados empezando en el objeto nodo que se especifique.
- *TraversalCriteria* asocian reglas para determinar que relación/nodo debería ser visitado.
- *Role* es derivada de CosRelationships::Role devuelve una vista de los papeles y de las relaciones sea en la forma de estructura o de un iterador.
- *EdgeIterator* permite iterar a través de las relaciones asociadas con un papel.

Contención y referencia son relaciones muy comunes. Consecuentemente, el servicio de relación provee un conjunto estandarizado de interfaces para ambas relaciones. Estas dos relaciones son derivadas de las interfaces del servicio de relación existente y definen tipos IDL de CORBA para todos los papeles y relaciones que son específicos a contención y referencia. Los atributos derivados permiten definir el grado de cada relación.

3.7 LOS SERVICIOS DE CORBA: GESTION DEL SISTEMA Y SEGURIDAD

Los servicios de seguridad, licencias, sincronización y actualización permiten gestionar un entorno de objetos distribuidos. El servicio de exteriorización permite escribir los contenidos de un objeto en un flujo de tal forma que se puede intercambiar o mover. El

servicio de propiedades permite asociar dinámicamente atributos con componentes empaquetados en tiempo de ejecución. Este es nuestro último conjunto de servicios que presento, la ubicación dentro de CORBA se aprecia en la figura 3.7.1.

3.7.1 El servicio de exteriorización de CORBA

El servicio de exteriorización define interfaces para exteriorizar un objeto a un flujo e interiorizar un objeto de un flujo. Un flujo es un área que mantiene datos que puede estar en memoria, en un fichero de disco, o a través de una red. Se exterioriza un objeto a un flujo para transportarlo a diferentes procesos, maquinas, o ORBs. Se interioriza un objeto cuando se requiere regresarlo a la vida en su nuevo destino. Este proceso de dos pasos permite exportar el objeto fuera de su entorno en el ORB.

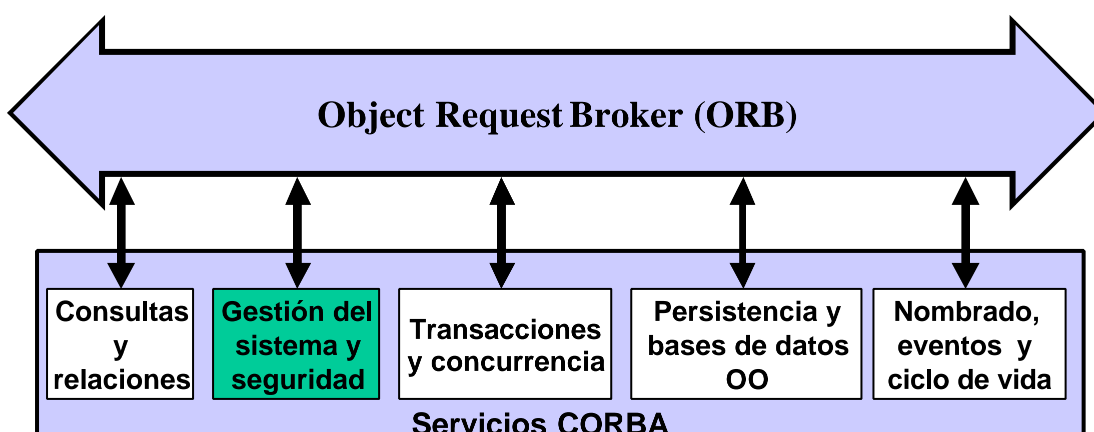


Figura 3.7.1 - Servicios de CORBA : Gestión del sistema y seguridad

Los flujos llegan a ser un medio de importación/exportación para los objetos. Permiten copiar y mover objetos y dejan también pasar objetos por valor en un parámetro.

Las interfaces del servicio de exteriorización son:

- *Stream* exterioriza e interioriza un objeto.
- *StreamFactory* crea un flujo.
- *FileStreamFactory* crea un flujo basado en un fichero.
- *StreamableFactory* crea un objeto que puede interiorizar un flujo.
- *StreamIO* lee y escribe los contenidos del flujo.
- *Streamaable* exterioriza e interioriza un objeto a/y desde un flujo.

El servicio de exteriorización también define un *Formato de Datos de Flujos Estándar* que permite intercambiar flujos entre distintas redes.

Esta representación de flujos estándar usa formatos de tipos de datos IDL autodescriptibles así como cabeceras que describen los tipos de objetos que están contenidos dentro del flujo.

3.7.2 El servicio de licencias de CORBA

Este servicio permite medir el uso de los componentes y cobrar de acuerdo a ello. Hace cumplir un amplio rango de opciones de licencias para satisfacer cualquier necesidad. Un componente notifica al servicio cuando desea ser controlado sin entrar en los detalles de como es llevado a cabo.

Todas las licencias deben tener inicio/duración y fecha de caducidad. Es posible también asignar algunas licencias a usuarios específicos, colecciones de usuarios y organizaciones. Finalmente el servicio de licencias debe ser un recurso del servidor muy seguro.

Este servicio consiste de dos interfaces que proveen todas las operaciones que un componente necesita para licenciarse/protegerse a sí mismo, son las siguientes :

- *LicenseServiceManager* sirve para localizar servicios de licencias que implementan políticas específicas. Regresa una referencia a un objeto para un servicio particular de licencia.
- *ProducerSpecificLicenseService* provee tres operaciones que hacen todo el trabajo. Iniciar el uso de la licencia, verificar su estado y terminar el uso de la misma.

Todas las operaciones entre el componente y el servicio son protegidas por un mecanismo de autenticación muy básico llamado desafío(challenge) que es un parámetro de entrada/salida que sirve como llave de autenticidad en el mensaje.

3.7.3 El servicio de propiedad de objetos de CORBA

El servicio de propiedad permite asociar dinámicamente atributos con componentes empaquetados. Se pueden definir estos atributos dinámicos (o propiedades) en tiempo de ejecución sin usar una IDL y luego asociarlos con un objeto que ya exista. Una vez que se definen estas propiedades, se les puede nombrar, fijar y obtener sus valores, determinar sus modos de acceso y eliminarlos.

El servicio de propiedad define cuatro modos de propiedad mutuamente exclusivos : normal, de sólo lectura, normal compuesto y de sólo lectura compuesto. De igual forma consta de seis interfaces, las dos principales son las siguientes :

- *PropertySet* define diez operaciones que permite definir, borrar, enumerar, y chequear la existencia de propiedades.
- *PropertySetDef* define ocho operaciones que permiten controlar y modificar los modos de operación.

El servicio de propiedad también provee interfaces que permiten manipular a las propiedades en grupos, factor que es indispensable en ambientes distribuidos.

3.7.4 El servicio de sincronización de objetos de CORBA

El servicio de sincronización de objetos es el que permite a un ORB mantener sincronizados los relojes de distintas máquinas distribuidas en una red con la finalidad de mantener la noción del tiempo y poder ejecutar eventos que tendrán que ocurrir en un sistema de objetos distribuido en un orden especifico.

3.7.5 El servicio de seguridad de objetos de CORBA

La seguridad es un servicio de objetos que afecta cada aspecto de un objeto distribuido. Siendo un aspecto de vital importancia, los ORBs con afán de controlarla usan autentificadores de terceros grupos de confiaza, que permiten a los objetos probar uno al otro, que ellos son quienes ellos dicen ser. Ambos grupos obtienen las palabras llave separadamente de un tercer grupo.

Un usuario es autentificado una vez por el ORB y le es dado un papel, selecciona derechos y seguridad. De la autenticación se genera un único identificador de verificación que no puede ser cambiado por un objeto ; solo el servidor de autenticación puede cambiarlo.

Una vez que los clientes son autentificados, los objetos del servidor son responsables de verificar que operaciones le son permitidas desempeñar a los clientes en la información que ellos tratan de acceder. Los servidores usan listas de control de accesos (ACLs) y una derivación llamada listas de capacidad para controlar el acceso a los usuarios. Las ACLs pueden ser asociadas con los recursos del ordenador. Es posible también que se desee el control de acceso a un nivel más bajo de granularidad haciendo a cada método de un objeto un recurso controlado.

Los servicios de auditoría permiten a los administradores del sistema monitorizar los eventos en el ORB, incluyendo intentos de acceso así como que servidores u objetos están siendo usados.

La información debe estar protegida contra corrupción o replica. Para lograr esto el ORB provee a quien envía de una prueba de entrega, y al receptor de una prueba de identificación de quien ha enviado. De igual forma los ORBs proveen de dos mecanismos que contribuyen a la seguridad que son : encriptación y verificación de sumas criptográficas.

3.7.6 El servicio de actualización de objetos de CORBA

El servicio actualización permite seguir diferentes versiones de los componentes y sus interfaces tal como se desarrollan. Éste servicio asegura que la instancia de un objeto usa una versión de implementación consistente.

Es posible encontrar la versión de un objeto por nombre. La versión esta directamente codificada en la referencia. Esto significa que se pueden invocar distintas versiones de un mismo objeto a la vez. La versión por defecto es siempre la última.

3.8 LOS OBJETOS DE NEGOCIOS (BUSINESS OBJECT)

Un objeto de negocios es un componente de nivel de aplicación que se puede usar en combinaciones impredecibles. Un objeto de negocios es por definición independiente de cualquier aplicación individual.

Los objetos de negocios serán usados para diseñar sistemas que imiten el proceso de negocios que ellos soportan. Estos objetos deben ser capaces de comunicarse uno con el otro en un nivel semántico.

Los objetos de negocios deben tener interfaces bien definidas de tal forma que puedan ser implementados independientemente, deben ser capaces de reconocer los eventos en su entorno, cambiar sus atributos, e interaccionar con otros objetos de negocios. Como cualquier objeto de CORBA, los objetos de negocios exponen sus interfaces a sus clientes vía IDL y se comunica con los otros objetos usando el ORB.

3.8.1 La anatomía de un objeto de negocios

De acuerdo con el modelo del Grupo de Interés Especial en Gestión de Objetos de Negocios de OMG (BOMSIG), un componente de objeto de negocios típico consiste de un objeto de negocios, uno o más objetos de presentación, y un objeto de procesos. Se describe cada tipo de objeto:

- *Objetos de negocios* encapsulan el almacenamiento, metadatos, concurrencia, reglas de negocios asociadas con una entidad de negocios activa.
- *Objetos de procesos de negocios* encapsulan la lógica del negocio al nivel de empresa. Los objetos de larga duración que involucran otros objetos de negocios son también controlados por estos objetos de procesos.
- *Objetos de presentación* representa el objeto visualmente hacia el usuario. Cada objeto de negocios pueden tener múltiples presentaciones para diversos propósitos.

Típicamente, en un objeto de negocios los objetos de presentación se encuentran distribuidos a través de múltiples clientes. El objeto de negocios y el objeto de procesos puede residir en uno o más servidores. Un objeto de negocios bien diseñado se auxilia de los servicios de CORBA, por ejemplo se puede usar el servicio de concurrencia y transacción para mantener la integridad de un objeto.

3.9 LAS FACILIDADES COMUNES DE CORBA

Las facilidades comunes de CORBA es el área más nueva de estandarización de OMG, ellas representan el más alto nivel de marcos de trabajo(frameworks) de componentes que completan la visión de CORBA para interoperabilidad.

Las facilidades comunes de CORBA son colecciones de jerarquías de clases bien definidas que proveen servicios directamente útiles a los objetos de negocios. Las dos categorías de facilidades comunes que son la horizontal y vertical definen las reglas de juego que son necesitadas por los componentes de negocios para colaborar efectivamente. Las facilidades verticales realizan la gestión de diversos aspectos mientras que las horizontales proveen IDLs definidas y marcos de trabajo para un segmento del mercado específico como salud o ventas.

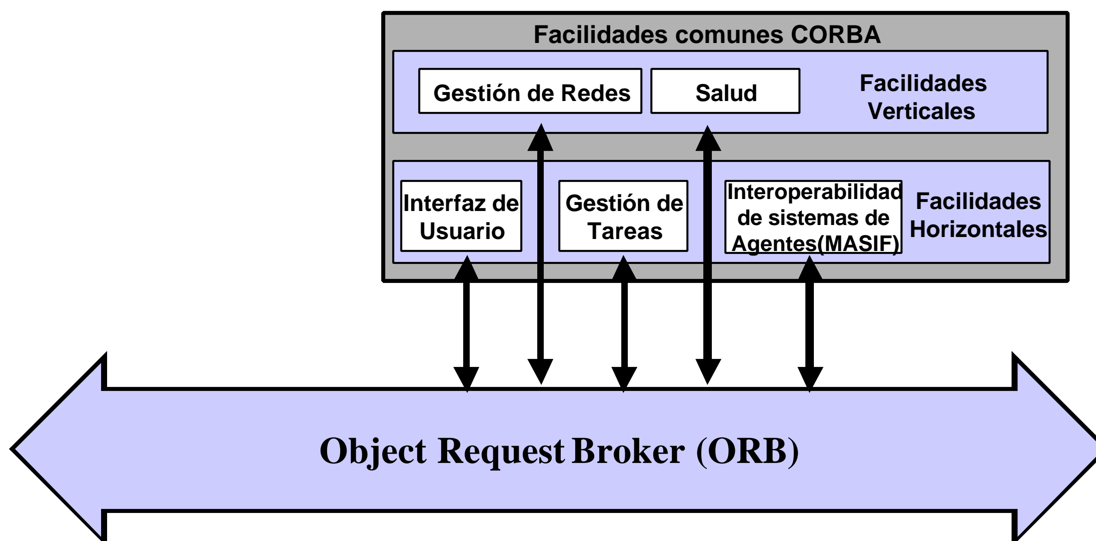


Figura 3.9.1 - Facilidades comunes de CORBA

3.9.1 Las facilidades comunes horizontales de CORBA

La facilidad común de Interfaz de usuario trabaja con la tecnología de documentos compuestos y con servicios de edición apropiados similares a los que provee OpenDoc y OLE. La idea es proveer un marco de trabajo para compartir y subdividir el desplegado de una ventana de tal forma que diferentes componentes puedan compartir el estado visual real de la pantalla en una forma integra.

Un componente debe ser capaz de incrustar otros componentes dentro de él y también debe ser capaz de editar sus contenidos en el lugar sin abandonar esa ventana. Esta facilidad trabaja con programación. Los lenguajes de programación usan mecanismos de enlace retardado lo que permite crear y unir programas en tiempo de ejecución a componentes en documentos compuestos. De igual forma usando un lenguaje de programación es posible invocar un servicio de CORBA IDL específico.

La facilidad común de gestión de la información incluye almacenamiento de documentos compuestos y facilidades para el intercambio de información. También define estándares que los componentes usan para codificar y representar sus datos, para definir e intercambiar metadatos, y para modelar información.

Los metadatos son particularmente importantes en un entorno de componentes distribuidos ya que son usados para describir servicios e información. La información en metadatos puede ser un suplemento al repositorio de interfaces describiendo la estructura y procedimientos de acceso a los datos. Ayuda a los componentes a buscar información efectivamente y a descubrir como son recuperados sus contenidos.

La facilidad común de gestión del sistema incluye interfaces y servicios para la gestión, instrumentación, configuración, instalación, operación y reparación de componentes de objetos distribuidos. La jerarquía de clases de gestión del sistema define las siguientes interfaces:

- *Instrumentation* permite colectar información acerca de la sobrecarga, responsabilidad, y consumo de recursos de un componente.
- *Data collection* reúne información acerca de eventos históricos relacionados con una componente.
- *Quality of service* selecciona el nivel de servicio que un componente provee en áreas como disponibilidad y desempeño.
- *Security* permite gestionar la seguridad del mismo sistema.
- *Event management* genera, registra, filtra y envía notificaciones de eventos a las aplicaciones en gestión.
- *Scheduling* programa tareas repetitivas y asocia gestores de eventos con eventos.
- *Instance tracking* asocian objetos con otros objetos gestionados que están sujetos a políticas comunes.

La facilidad común de gestión de tareas provee un esquema de trabajo para la gestión de flujos de trabajo, grandes transacciones, agentes, programación, reglas y automatización de tareas. Incluye una facilidad semántica de mensajes usada para comunicar peticiones orientadas a tareas.

Los agentes consisten en objetos de información y un programa asociado que sabe que hacer con la información y como trabajar con el entorno, existen agentes estáticos y móviles. La infraestructura de agentes de CORBA provee capacidades para que los agentes publiquen sus servicios y para que los subscriptores se registren a estos servicios.

La gestión de reglas provee facilidades para especificar reglas evento/condición/acción en un lenguaje declarativo y para gestionar y ejecutar estas reglas. OMG provee un lenguaje de especificación de reglas basados en IDLs.

La automatización es la tecnología usada por sistemas de programación o cualquier cliente que necesite dinámicamente acceder a las funciones de los componentes o datos contenidos. Se puede usar la automatización para gestionar objetos visibles por el usuario que son más detallados generalmente que los objetos típicos de un ORB. La automatización requiere enlaces retardados, el cual puede ser proveído usando la invocación de métodos dinámicos de CORBA y la interfaz de esqueletos dinámica.

3.9.2 Las facilidades comunes verticales de CORBA

Las facilidades verticales proveerán interfaces IDL definidas estándares para la interoperabilidad de componentes para segmentos del mercado tales como salud, ventas y finanzas. Algunos de los marcos de trabajo que están en desarrollo son los siguientes.

- *CORBA para la replica de documentos*. Incluye almacenamientos de documentos imagen que pueden mover y copiar grandes objetos binarios, indexarlos, etc.
- *CORBA para supercarreteras de información*. Este esquema usa a CORBA como un medio para un gran flujo de información.
- *CORBA para la manufactura integrada por computadora.*
- *CORBA para simulaciones distribuidas.* Este esquema puede ser usado para la simulación de control de tráfico aéreo, escenarios de negocios, etc.
- *CORBA para la exploración de gas y petróleo.*
- *CORBA para contabilidad.* Provee un sistema de contabilidad distribuido que incluye transacciones de negocios, cambio de monedas, ventas, etc.

Capítulo IV
OPENDOC

Opendoc es un caso excelente para el estudio de componentes. Los componentes de Opendoc pueden compartir una ventana en la pantalla, intercambiar contenidos basados en mensajes semánticos y almacenar sus contenidos dentro del mismo fichero del documento. Una segunda mejor característica es que las partes son codificables lo que le permite combinar partes de diferentes proveedores para lograr una solución particular. Analizaremos con más detalle el modelo de objetos de OpenDoc el cual puede apreciarse en la figura 4.1:

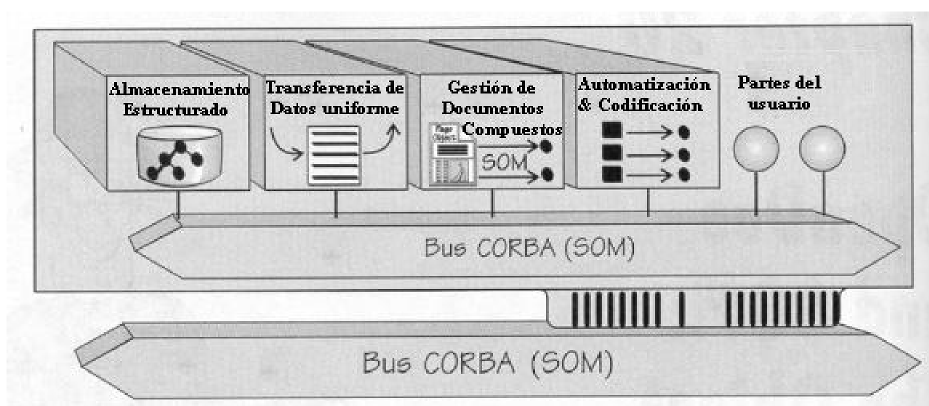


Figura 4.1 - El modelo de objetos de OpenDoc.

4.1 OPENDOC : EL MODELO DEL OBJETO

OpenDoc es un conjunto de librerías de clases con interfaces IDL definidas independientes de la plataforma [ORFA96]. Sus interfaces se compilan separadamente de una implementación de un objeto usando esqueletos SOM compatibles con CORBA. OpenDoc instancia un conjunto de objetos CORBA, usa estándares de despacho distribuidos de un lenguaje neutral de SOM para todo su sistema. Los Objetos SOM son enlazados dinámicamente, permitiendo a las nuevas partes ser añadidas a los documentos existentes en cualquier momento.

IDL y SOM hacen posible para los editores de partes el comunicarse consistentemente y pasar parámetros con diferentes compiladores y en diferentes lenguajes de programación.

Entre otras cosas, SOM provee una tecnología de empaquetado de componentes. En el caso de OpenDoc, permite a los vendedores de software empaquetar sus editores de partes en librerías de clases binarias y llevarlas en DLLs.

Adicionalmente, SOM soporta herencia, lo que significa que los programadores pueden crear subclases de las partes de OpenDoc y entonces rehusar o sobrescribir las implementaciones de sus métodos entregadas en DLL binarios. El IDL compatible con CORBA de SOM hace posible que los editores de partes de OpenDoc sean distribuidos en un formato binario, lo cual significa que el código fuente no tiene que ser proveído. Si una clase de SOM es modificada y recompilada, no requiere la recompilación de todas sus clases derivadas y sus clientes.

4.1.1 El modelo de programación de OpenDoc.

OpenDoc introduce una disciplina de componentes para el diseño de interfaces de usuarios. Una parte de OpenDoc consiste de datos y de un editor de partes que manipula esos datos.

Para crear un editor de partes de OpenDoc, se debe crear una subclase de *ODPart* una clase abstracta que define su comportamiento. Como mínimo una parte deber ser capaz de reservar almacenamiento para sus datos persistentes, inicializar sus datos de un almacenamiento persistente, dibujar sus contenidos dentro de un área proveída por su contenedor, gestionar eventos, y exteriorizar sus datos a un almacenamiento persistente Bento.

Para programar una parte, primero se define la interfaz en un fichero fuente IDL, entonces se ejecuta el precompilador de SOM en el fichero, que produce un esqueleto de implementación para la clase. Se añade el cuerpo de la implementación al esqueleto. Entonces se compila la clase y se crea la parte DLL.

El ejecutor de OpenDoc es una colección de objetos. Un editor de partes interactúa con estos objetos invocando métodos de ellos siempre que necesite un servicio de OpenDoc. OpenDoc no hace mucho uso de la herencia y prácticamente no hay herencia múltiple en cualquier parte de su jerarquía de clases. La razón es que OpenDoc realmente encapsula muchos grupos no relacionados de servicios. Para proveer de funcionalidad al editor de partes se debe personalizar la clase ODPart a través de herencia.

OpenDoc debe proveer un espacio de direcciones compartido que los editores puedan usar para manipular el contenido de los documentos y obtener servicios; este entorno lo proporciona a través de un conjunto de instancias de objetos que pertenecen a un proceso de *shell* para cada documento.

OpenDoc crea una instancia del entorno de *shell* cada vez que un documento es abierto. El *shell* entonces inicializa un objeto *ODSession*, el objeto sesión en turno, instancia objetos de servicios de OpenDoc y mantiene referencias a ellos. Las partes encuentran y obtienen acceso a la mayoría del entorno de OpenDoc vía el objeto de sesión. *ODSession* conoce por nombre o más precisamente por referencia cada objeto de servicio de OpenDoc.

OpenDoc es un sistema orientado a objetos, la programación con OpenDoc no es muy propietaria por:

- Se dibujan las partes usando las APIs de la plataforma nativa.
- Se almacenan datos usando flujos ordinarios.
- Se procesan los eventos de OpenDoc usando la cola de eventos de la plataforma.
- Se usan los modelos de memoria existentes.

OpenDoc fue diseñado para proveer a través de la plataforma una arquitectura de documentos compuestos sobre una plataforma específica de sistemas gráficos e interfaces de dibujo. En adición, OpenDoc fue diseñado para comunicaciones entre partes.

Las arquitecturas de documentos compuestos introducen nuevas posibilidades en el diseño de esquemas de trabajo (*framework*) porque la unidad de construcción es un componente granular. La comunidad de OpenDoc tiene dos esquemas de trabajo de donde escoger:

- *Esquema de trabajo de desarrollo de OpenDoc* proporciona a los programadores funciones del editor de partes preconstruidas de donde se pueden crear subclases para su uso particular.
- *Esquema de trabajo de OpenDoc* de IBM construye una librería de clases a través de la plataforma que proporciona funciones como colecciones y bases de datos.

El éxito de OpenDoc dependerá de la calidad de las herramientas que los vendedores proporcionen para hacer más fácil la construcción y ensamblado de partes. OpenDoc tiene el potencial de revolucionar el mercado de herramientas para las aplicaciones.

4.2 OPENDOC: EL MODELO DE DOCUMENTOS COMPUESTOS

En un documento compuesto, el documento posee el proceso y es responsable por la interacción con el sistema operativo. OpenDoc le llama a este proceso poseído por el documento *shell* de documento. Este debe proveer un entorno para el despacho de eventos del sistema operativo al gestor de partes apropiado.

4.2.1 Creación del entorno de OpenDoc

Opendoc crea una instancia del shell de documento cada vez que un documento es abierto. Este crea e inicializa un objeto *ODSession*, enlaza y carga los editores de partes, permite a los editores de partes acceder a variables globales así como recibe los eventos de los usuarios y los pasa al objeto *ODDispatcher* de OpenDoc.

Todos los objetos de OpenDoc son instanciados a través de una fábrica de métodos. Las fábricas ayudan también a la recolección de basura. Una parte es un objeto porque encapsula estado y comportamiento. La parte de datos provee el estado y el editor de partes provee el comportamiento.

4.2.2 Gestión de la estructura de documentos.

OpenDoc usa cuatro ideas simples para crear estructuras de documentos compuestos: documentos, sus partes, sus marcos y el código del editor de partes que los manipula. Un documento de OpenDoc es una jerarquía persistente de partes incrustadas dentro de partes. OpenDoc también usa una jerarquía separada pero esencial para hacer el desplegado de elementos llamados facetas.

En la cima de la jerarquía de partes, esta una parte raíz. Esta es la parte original que controla el esquema básico del documento, tamaño de página, etc. Los contenedores son partes que incrustan a otras partes y que pueden también estar incrustadas en otros contenedores. Los objetos *frame* son usados para las negociaciones de espacio entre las partes contenidas y las incrustadas.

Las facetas representan a un *frame* visible en tiempo de ejecución. Las facetas sólo son requeridas por aquellos *frames* que sean visibles en un momento dado. En contraste los *frames* deben existir para todas las partes incrustadas en el documento. La siguiente pieza son los lienzos (*canvas*) que es donde las facetas de una parte se reproducen a sí mismas. Una faceta esta asociada con un lienzo de dibujo específico. Finalmente el objeto forma (*shape*) representa el espacio y coordenadas de una figura de dos dimensiones en un lienzo. Una figura puede tener cualquier tipo de contorno, aún irregulares.

Veamos pues la forma de interacción de estos componentes para recomponer un documento de su almacenamiento. Cuando se abre un documento OpenDoc insta un nuevo proceso de *shell* de documento el cual lee el documento a memoria. El *shell* usa la ventana de estado de información salvada para reconstruir las ventanas tal como estaban en la pantalla la última vez que fueron grabadas.

Una vez que el *shell* tiene la ventana, lee el *frame* raíz y construye la faceta raíz. El *frame* raíz lee la parte raíz; la parte raíz lee sus *frames* incrustados y construye las facetas para ellos. Los *frames* incrustados leen sus propias partes y a su vez sus *frames* incrustados y crean facetas para ellos. Después de que todos los objetos son leídos, el *shell* pide a cada faceta que se dibuje a sí misma. El editor de partes asociado con el *frame* de la faceta dibuja la parte visible contenida en la faceta. De igual forma si el

estado fue salvado, el editor de partes puede activar la paleta o menú que estuviera activo la última vez que se salvo.

Como se aprecia el dibujar el documento es un esfuerzo conjunto. Cada parte contenedora controla el tamaño, forma y posición de las partes incrustadas en él. Las partes de OpenDoc pueden desplegarse en diferentes maneras en *frames* separados, o en diferentes maneras en el mismo *frame* en tiempos diferentes. OpenDoc soporta la edición en la misma ventana donde se encuentra la parte.

Los controles en OpenDoc proveen elementos de interfaz gráfica que incluyen botones, barras de herramientas y campos.

4.2.3 Distribución de eventos y arbitraje

Los editores de partes interaccionan con los usuarios respondiendo a sus eventos. Estos eventos son enviados por el gestor de colas de eventos del sistema operativo nativo. El *shell* recibe los eventos de los usuarios, gestiona la mayoría de los comandos de menú de documento y el resto lo pasa al despachador quien coopera con un objeto de tipo *ODArbitrator* para distribuir el evento a la parte del documento apropiada.

Opendoc permite que múltiples partes trabajen concurrentemente dentro del mismo documento y usa el modelo de activación dentro/fuera que permite la edición de cualquier parte visible sin previa activación.

4.3 OPENDOC: BENTO Y UNIDADES DE ALMACENAMIENTO

Bento el sistema de almacenamiento de OpenDoc provee un mecanismo de almacenamiento persistente que permite a varios editores de partes compartir un fichero individual de un documento. Bento permite que nuevos tipos de información sean definidos sin destruir las estructuras existentes. Bento fue diseñado para gestionar requerimientos de multimedia en tiempo real para que ficheros de animación y sonido puedan ser ejecutados sin interrupciones.

Bento puede ser usado para intercambiar datos, partes, grupos de partes y documentos completos usando las facilidades de transferencia de datos de OpenDoc. Adicionalmente, Bento provee estructuras para mantener el registro de información tales como partes y sus relaciones de almacenamiento de una con la otra.

4.3.1 El sistema de almacenamiento de Bento

Los objetos de Bento pueden ser pequeños o enormes, pueden ser ficheros o estructuras complejas. Bento permite que se almacenen representaciones múltiples del mismo contenido dentro de cada propiedad.

Bento no es un ODBMS, él gestiona un sistema de ficheros estructurados y referencias entre partidas de datos. Bento subdivide un fichero en un sistema de elementos estructurados; cada elemento puede contener muchos flujos de datos. Esto permite a cada parte tener su propio flujo de almacenamiento. Es como si existiera un sistema de ficheros dentro de cada fichero.

Los objetos que son de mayor interés para los usuarios y programadores de OpenDoc son los documentos, planos(*drafts*) y unidades de almacenamiento. Un objeto documento contiene datos para todas sus partes, sus enlaces e información a cerca de *frames* y datos incrustados. Los planos son vistas que representan el estado de un documento en un punto y tiempo dado. Siempre existe al menos un plano dentro de cada documento.

4.3.2 Unidades de almacenamiento de OpenDoc

Una unidad de almacenamiento es donde los datos de una parte vive. Las unidades de almacenamiento son la unidad básica de almacenamiento persistente para una parte. Una unidad de almacenamiento contiene una lista de propiedades; cada propiedad tiene un nombre único dentro de la unidad de almacenamiento. Una propiedad puede tener uno o más valores. Los valores pueden ser flujos de bytes o pueden tener múltiples tipos de datos.

Los documentos de Bento consisten de múltiples planos. Cada plano contiene diversas unidades de almacenamiento; estas unidades a su vez pueden contener propiedades y sus valores. Los valores pueden incluir referencias a otras unidades de almacenamiento. Es posible leer valores o datos selectivamente de unidades de almacenamiento sin tener que leer el fichero completo o la parte de datos.

Una referencia persistente es un número único de 32 bits que almacenado dentro de una unidad de almacenamiento que apunta a otra unidad de almacenamiento en el mismo documento. OpenDoc usa estas referencias para mantener relaciones en su jerarquía de partes.

OpenDoc especifica un conjunto de propiedades estándar que todos los editores de partes pueden reconocer. Esto hace más fácil inspeccionar los contenidos de las unidades de almacenamiento a través de las partes, documentos y plataformas.

4.4 OPENDOC : TRASFERENCIA DE DATOS UNIFORME

La transferencia de datos uniforme basada en Bento de Opendoc permite a las partes que no saben nada unas de las otras intercambiar datos, partes y múltiples partes. Los usuarios finales pueden orquestar los intercambios de datos y usarlos para poner cualquier tipo de información en un documento. Opendoc tiene tres mecanismos para esto:

- *Mover y soltar (*Drag and drop) permite manipular datos y partes directamente, es posible moverlos dentro de un mismo documento o entre documentos diferentes.
- *Portapapeles* permite el intercambio de datos a través de un recipiente intermedio, se usan los comandos cortar, copiar y pegar.
- *Enlazado* permite copiar datos de las mismas formas que las anteriores, la diferencia es que los enlaces son mantenidos entre las partes origen y destino. Cuando las partes o datos del origen cambian, los enlaces del destino son automáticamente refrescados para reflejar los cambios. El objeto de enlace origen es almacenado en un directorio compartido de tal forma que los subscriptores puedan obtener sus datos aunque la parte origen no este corriendo. Adicionalmente a la copia de datos, el objeto fuente mantiene enlaces persistentes a todas las partes destino que suscribe el contenido del origen.

Cuando se copia un componente (una parte), se hace una copia profunda del objeto, es decir, son copiados todos los objetos que éste referencia, más los objetos que ellos hagan referencia y así sucesivamente. Adicionalmente, OpenDoc usa una forma inteligente de soltar (*drop*) y pegar donde se puede decidir si incrustar el dato como una parte separada o incorporarla como datos contenidos en la parte destino. Las partes de igual forma pueden tener una lista de inclusión que especifica los tipos de partes que pueden ser incrustadas dentro de ellas.

4.5 OPENDOC : AUTOMATIZACION Y EVENTOS SEMANTICOS

La arquitectura abierta de programación (*OSA de Open Scripting Arquitecture*) es una tecnología de habilitación para un nuevo paradigma de programación que ve a las aplicaciones como colecciones de partes componentes. Estas partes puede ser enlazadas todas juntas vía programas escritos en una variedad de lenguajes de programación.

La OSA define cómo contenedores y partes se comunican unos con otros usando un registro de eventos. Los tres elementos que proveen la automatización y servicios de programación de opendoc son:

- *Las extensiones* son usadas por editores de partes para crear y publicar servicios añadidos. Los clientes pueden dinámicamente preguntar a los servidores de partes por las extensiones que ellos soportan.
- *Los eventos semánticos* son usados para que los usuarios puedan actuar no sólo con las partes sino también en el contenido de los objetos dentro de las partes. Los

eventos semánticos son los medios por los cuales los sistemas de programación manipulan los contenidos de una parte programable.
- *La programación abierta* provee el tercer elemento del modelo OSA. Un lenguaje de programación debe ser capaz de traducir código en eventos semánticos. Adicionalmente los sistemas de programación deben proveer una interfaz genérica que permita a las partes manipular los códigos, almacenarlos, y unirlos a eventos.

4.5.1 El nuevo mundo de programación de OpenDoc.

La programación proporciona las bases para la tecnología de componentes. Los agentes especialmente los móviles, parecen ser autosuficientes. Los mejores agentes llevan su entorno de ejecución con ellos mismos a donde quiera que vayan.

Usando códigos los usuarios experimentados pueden gestionar componentes individuales o ensamblar componentes a través de redes y sistemas operativos. Los códigos pueden ser también usados para extender el comportamiento individual de un componente.

Se pueden usar programas para crear documentos móviles muy inteligentes. Por ejemplo un programa de verificación de acceso puede ser ejecutado cada vez que se abra un documento.

Después de que un documento es abierto, los programas unidos a las partes pueden operar en los contenidos del documento y ejecutar transacciones contra servidores remotos o partes locales. Los programas pueden operar directamente en los contenidos de una parte a través de eventos semánticos.

Los agentes inteligentes usarán programas para unir aplicaciones ensambladas creadas por la unión de varias partes vía programas. Incrementalmente, los programas llegarán a ser el paradigma predominante para orquestar colaboraciones de intercomponentes tanto en el cliente como en el servidor y entre las partes del cliente y del servidor.

4.5.2 El mecanismo de extensión de OpenDoc

El mecanismo de extensión de OpenDoc permite que las partes colaboren, compartan información o que manipulen los contenidos de otra. Este mecanismo es usado para desarrollar series de partes que colaboran entre sí. La idea es que un editor de partes cree objetos de extensión asociados que implementen las interfaces adicionales.

Las interfaces de extensión permiten a las partes incrementar sus capacidades extendiendo sus interfaces de programación. En tiempo de ejecución el editor de partes registra las extensiones con Opendoc para hacerlas disponibles al mundo exterior.

Los editores de partes extensibles crean y borran sus propios objetos de extensión. Ellos también gestionan su almacenamiento.

4.5.3 Los eventos semánticos

Los eventos semánticos representan las acciones de más alto nivel en el nivel de aplicación. Éstos pueden representar acciones tales como la apertura o cierre de un documento; sin embargo la mayoría de los eventos semánticos son usados para manipular los contenidos de una parte.

Los eventos semánticos proveen una forma de mensajes que las partes pueden usar para requerir servicios e información dinámicamente una de otra. Un cliente inicia un evento semántico; un servidor lo ejecuta. El cliente y el servidor pueden residir en la misma máquina.

El modelo especificador de objetos proporciona una manera para describir un arreglo jerárquico de los contenidos de los objetos cuya naturaleza exacta depende de los contenidos de las partes. Un objeto especificador puede hacer referencia a cualquiera de los siguientes objetos: una aplicación, un documento, una parte dentro de un documento, una parte incrustada dentro de una parte o algún elemento contenido dentro de una parte.

Las partes pueden enviar eventos semánticos a sus partes contenidas, a partes incrustadas, etc. Los eventos semánticos están diseñados para poder enviar mensajes entre partes a través de documentos y a través de redes.

Para ser programable, una parte debe ser capaz de al menos recibir eventos semánticos, ejecutarlos, y contestarlos; no tiene que enviarlos.

4.5.4 Programación y automatización

Un sistema de programación permite a un usuario experimentado diseñar, adjuntar, y ejecutar código para generar los eventos semánticos que una parte programable recibe. Opendoc soporta cualquier lenguaje de programación.

A causa de la flexibilidad de OSA, los editores de partes pueden proveer diferentes niveles de programación.

- *Capacidad de codificación:* es la forma más simple de soporte programacional. Para ser programable, un editor de partes solo necesita publicar una descripción de sus objetos contenidos y sus operaciones. Para que una parte sea completamente programable, los eventos semánticos deben ser capaces de invocar cualquier acción y los usuarios deben ser capaces de desempeñarla.
- *Asociación a eventos:* es el siguiente nivel de soporte programacional. Permite a los usuarios unir programas a eventos que una parte programable recibe o genera. Los eventos incluyen acciones en las interfaces de usuario, edición, apertura, salvado, y cambios de mayor estado dentro del contenido de una parte.
- Capacidad de reenvío: es la forma extrema de la asociación a eventos. Un editor de partes con capacidad de grabado crea un embotellamiento que intercepta cualquier evento y llamada y los reenvía a sí mismo como eventos semánticos. El código del

editor de partes es dividido en un elemento de interfaz que acepta los eventos y el código que realmente hace el trabajo.

Una parte de OpenDoc puede conectarse a uno o mas sistemas de programación que usen uno o más lenguajes de programación a través de una objeto de la clase *ODSAObject.* Antes de que una parte pueda manipular y ejecutar programas, primero debe conectarse a un sistema de programación de OSA.

Opendoc usa el repositorio de interfaces de CORBA para registrar los componentes de programación y sus propiedades.

Capítulo V
COM (COMPONENT OBJECT MODEL)

La actual implementación del modelo de objetos componentes (COM) es una versión para una máquina individual de un bus de objetos distribuidos. El gestor de COM sólo trabaja con objetos OLE. El COM distribuido (DCOM), será simplemente una extensión de la implementación de COM de hoy en día y proveerá transparencia local y remota sin modificar los objetos OLE actuales.

COM consta de dos elementos: el bus de objetos que claramente pertenece a COM y los servicios de objetos que serán todos los documentos no compuestos que son elementos de OLE.

La figura 5.1 muestra en detalle los elementos así como la interacción y flujo de datos en el modelo de COM, los cuales veremos en las siguientes secciones.

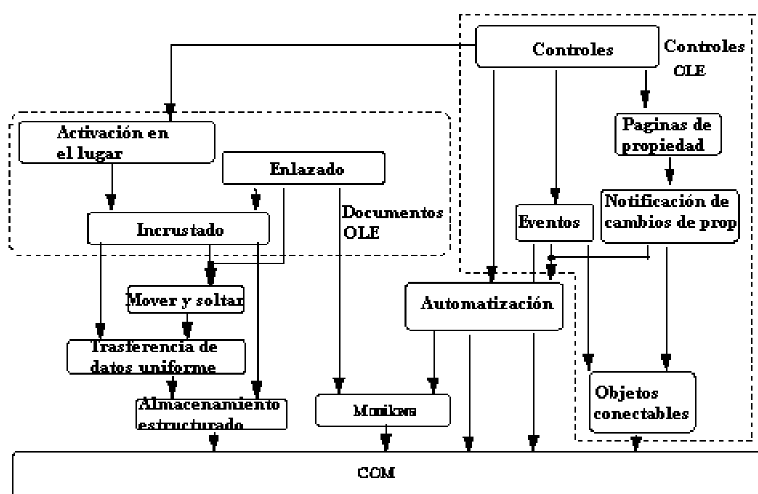


Figura 5.1 – Elementos de COM

5.1 COM : EL BUS DE OBJETOS DE OLE

COM separa la interfaz del objeto de su implementación y requiere que todas las interfaces sean declaradas usando un lenguaje de definición de interfaces(IDL). El IDL de Microsoft esta basado en DCE.

COM no soporta herencia múltiple en las IDLs especificadas. Sin embargo, un componente COM puede soportar múltiples interfaces y lograr el rehuso encapsulando las interfaces de los componentes más internos y presentándolos al cliente. Es decir se logra el rehuso vía contención y agregación más que herencia.

Un objeto COM no es un objeto en el sentido de Orientación a objetos. Las interfaces de COM no tienen estado y no pueden ser instanciadas para crear un objeto único. Una interfaz es simplemente un grupo de funciones relacionadas. A los clientes de COM les es dado un puntero para acceder a las funciones de una interfaz.
COM provee interfaces estáticas y dinámicas para la invocación de métodos. La librería *Type* es la versión de COM de un repositorio de interfaces. Los precompiladores de COM pueden incrementar la librería *Type* con las descripciones de objetos ODL-definidas (ODL de *Object Description Lenguage)* incluyendo interfaces y parámetros. Los clientes pueden consultar esta librería para descubrir las interfaces que un objeto soporta. COM también provee un registro y algunos servicios de búsqueda de objetos similares a un repositorio de interfaces de CORBA.

A diferencia de CORBA, el bus de COM no se extiende dentro de redes. Es una implementación de un ORB para una máquina individual.

5.1.1 El estilo de interfaces de COM

Una interfaz de COM es una colección de métodos. Ésta define funciones independientemente de su implementación. Los clientes de COM interactuan unos con otros y con el sistema llamando las funciones miembro de sus interfaces. Una interfaz COM es independiente del lenguaje y puede llamar funciones a través de un espacio de direcciones(pero no a través de redes). Sin embargo a diferencia de CORBA, COM no provee enlaces del lenguaje de alto nivel.

Una interfaz de COM es definida como una API binaria de bajo nivel en una tabla de punteros. Los clientes de COM para acceder a una interfaz usan punteros a un arreglo de punteros de funciones. En tiempo de ejecución, cada interfaz es conocida por su identificador de interfaz (IID). Un IID es un identificador global único (GUID) generado por COM para sus interfaces. Los GUIDs son IDs de 128 bits y son generadas de forma única llamando la función *CoCreateGuid* de la API de COM. La IID permite a un cliente preguntar si un objeto soporta su interfaz.

5.1.2 Qué es un objeto COM

Un objeto COM también conocido como objeto OLE es un componente que soporta una o más interfaces. Una interfaz de COM se refiere a un grupo predefinido de funciones relacionadas. Una clase de COM implementa una o más interfaces y es definida por un identificador de clase único de 128 bits (CLSID). Un objeto COM es una instancia de una clase en tiempo de ejecución.

Los clientes siempre trabajan con los objetos de COM a través de los punteros a sus interfaces; ellos nunca acceden directamente al objeto mismo. Note que una interfaz no es un objeto en el sentido clásico. Los objetos de COM no soportan IDs únicos de objetos.

Todos los objetos de COM deben implementar la interfaz *IUnknown* a través del cual el cliente puede controlar el tiempo de vida de un objeto. Un cliente puede preguntar a un objeto que interfaz soporta y obtener punteros a ellos.

5.1.3 Qué es un servidor COM

Un servidor COM es una pieza de código un DLL o un EXE que almacena una o más clases de objetos con su propia CLSID. Cuando un cliente pregunta por un objeto de una CLSID dada, COM carga el código del servidor y pide crear un objeto de aquella clase. El servidor debe proveer una fabrica de clases para crear un nuevo objeto. Una vez que un objeto es creado, un puntero a su interfaz primaria es regresado al cliente.

Así pues un servidor COM provee la estructura necesaria para que un objeto pueda hacerse disponible a sus clientes. Más específicamente un servidor COM debe:

- *Implementar una interfaz para una fabrica de clases.*
- *Registrar las clases que soporta.* El servidor debe registrar una CLSID para cada clase que soporta en el repositorio de Windows.
- *Inicializar la librería de COM.*
- *Verificar que la librería es de una versión compatible.*
- *Proveer un mecanismo de descarga del servidor.*
- *Desinicializar la librería de COM.*

La implementación del servidor, incluyendo el registro, la fabrica de clases, y el mecanismo de descarga diferirá dependiendo sí el servidor esta empaquetado como un DLL o un EXE.

COM define tres tipos de servidores:

- *Los servidores en proceso* se ejecutan en el mismo espacio de procesos como sus clientes.

- *Los servidores locales* se ejecutan en procesos separados de sus clientes en la misma máquina.
- *Los servidores remotos* ejecutan un proceso separado en una máquina remota y, posiblemente, sistema operativo diferente.

COM habilita a los clientes a que se comuniquen transparentemente con objetos servidores sin importar donde estén corriendo estos objetos. Desde el punto de vista de un cliente, todos los objetos en los servidores son accedidos a través de los punteros a las interfaces. Si un objeto esta en proceso, la llamada lo accede directamente. Si un objeto esta fuera de proceso, entonces la llamada primero a un objeto *proxy* proporcionado por COM mismo. Este objeto genera la llamada del procedimiento remoto apropiada a los otros procesos o a las otras máquinas.

Desde el punto de vista del servidor, todas las llamadas a las funciones de una interfaz de un objeto son hechas a través de un puntero a esa interfaz. Si el objeto esta en proceso, quien llama es el cliente mismo. De otra manera, quien llama es un objeto *stub* proveído por COM que recoge la llamada del procedimiento remoto del *proxy* en el proceso cliente y lo convierte en una llamada a una interfaz en el objeto servidor.

La librería de COM provee *proxies* y *stubs* para todas las interfaces estándar predefinidas de COM y OLE. Para desarrollar interfaces propias, los programadores necesitan crear un fichero que describa los métodos de la interfaz y sus argumentos usando el lenguaje de definición de interfaces (IDL) de COM. Ellos deben correr este fichero a través de un compilador MIDL de Microsoft para crear los *proxies* del cliente, los *stubs* del servidor, y el código que gestiona el paso de parámetros entre ellos[3].

5.2 LOS SERVICIOS DE OBJETOS

Cada servicio de objetos que no corresponde a documentos compuestos pertenecerá a COM. Los servicios tradicionales de COM incluyen negociaciones de interfaces, gestión del ciclo de vida a través de conteo de referencias y fabricas, licencias de componentes y el servicio de eventos. Monikers, objetos de datos, y estructuras de almacenamiento eventualmente pertenecerán a COM.

La interfaz *IUnknown* es el corazón de COM. Es usada para las negociaciones de interfaces en tiempo de ejecución, gestión del ciclo de vida y agregación. Cada interfaz COM y OLE debe implementar las tres funciones miembro de *IUnknown*: *QueryInterface, AddRef, y Release. IUnknown* es una interfaz base de las cuales todas las otras interfaces heredan estas tres funciones y las implementan de una forma polimorfa.

Cuando un cliente obtiene acceso inicialmente a un objeto, por cualquier medio, al cliente le es dado uno y solo un puntero a la interfaz en respuesta. El resto de las interfaces son obtenidas llamando *QueryInterface*. Esta función permite a los clientes descubrir en tiempo de ejecución si una interfaz especificada por un IID es soportada

[3] El proceso es análogo al que se realiza en CORBA para generar los esqueletos del cliente y el servidor.

por un objeto componente. El objeto puede rechazar el servicio a un cliente al no regresar un puntero, esto sucede si la interfaz no es soportada.
Debido a que COM no soporta herencia múltiple, un componente de COM debe ser capaz de agregar interfaces múltiples para lograr un efecto similar. *QueryInterface* también permite a los componentes añadir nuevas interfaces y a los clientes descubrirlas. Esto permite que nuevas interfaces sean añadidas a una clase de un objeto tal como su función se desarrolla.

5.2.1 Gestión del Ciclo de Vida:

OpenDoc y COM utilizan el mismo mecanismo. La solución está basada en mandar a todos los objetos mantener contadores de referencias y mandarles borrarse a sí mismos cuando ya no se encuentren en uso. El objeto sabrá que no está en uso cuando su contador de referencias llegue a cero.

COM hace el conteo de referencias ubicuo al añadir esta función a *IUnknown* a través de las funciones miembro *AddRef* y *Release*. La función *AddRef* de la interfaz es llamada cuando un cliente solicita un puntero a esta interfaz o le pasa una referencia. Una función *Realese* de una interfaz es llamada cuando un cliente no necesita más esa interfaz.

5.2.2 Creación de objetos y licencias

Cada objeto COM debe implementar una fábrica de clases que un servidor COM pueda invocar para crear instancias de aquella clase. Una fabrica de clases de COM es una implementación de las interfaces *IClassFactory* o *IClassFactory2*. La primera es una extensión de la segunda que refuerza el proceso de licencia en un objeto en el tiempo de creación. La interfaz *IClassFactory2* implementa dos funciones para la creación de nuevas instancias de una clase que son: *CreateInstance* y *CreateInstanceLic*.

COM considera que una maquina esta completamente autorizada cuando un fichero de licencias esta instalado en esa máquina. De otra manera, el cliente debe proveer una clave de licencia especial cuando éste crea una instancia del componente. Si la clave de licencia no es proveída, el servidor no instanciará el componente. El fichero provee un acceso global al componente en la máquina, mientras que la clave de licencia es un permiso específico para usar el componente en otra máquina. La implementación *IclassFactory2* validará la clave o fichero de licencia antes de crear una instancia del componente.

5.2.3 El servicio de Eventos de COM

Los objetos conectables son la versión de COM de un servicio de eventos estándar. Un objeto COM puede soportar interfaces ODL definidas tanto de entrada como de salida.

Estas interfaces proveen una forma estándar para definir eventos y sus parámetros. Los objetos que soportan estas interfaces son llamados conectables o fuentes. COM también provee interfaces llamadas sumideros (*sinks*) que permite a los objetos subscribirse a estos eventos de salida.

Los sumideros y las fuentes permiten a los clientes llegar a ser servidores y viceversa. Los eventos mencionados le dicen a cualquier sumidero que este escuchando que algo interesante acaba de sucederle al objeto. En COM, los productores y consumidores de eventos están directamente conectados a través de sumideros y fuentes.

5.2.4 El estilo de herencia de COM: Agregación y contención

A diferencia de CORBA, COM no soporta herencia múltiple[4]. Sin embargo un componente de COM puede soportar interfaces múltiples. OLE soporta dos métodos de encapsulación que es el medio a través del cual logra la herencia:

- *En Contención/delegación*, el objeto más externo debe ejecutar otra vez las invocaciones de métodos que recibe en nombre de sus objetos más internos.
- *En agregación* el objeto más externo expone las interfaces de los objetos más internos como propias.

Microsoft cree que la agregación y la contención proveen todo el rehuso que es requerido en un entorno distribuido.

5.3 OLE: AUTOMATIZACION, PROGRAMACION Y LIBRERIAS DE TIPOS

La automatización OLE permite a los programas del cliente invocar dinámicamente métodos que manipulan los contenidos de los objetos programables. La automatización OLE no es específica a ningún lenguaje de programación.

Los servicios de automatización usan el modelo COM de OLE, pero éstos pueden ser implementados independientemente del resto de OLE. Usando la automatización OLE se pueden crear aplicaciones que exponen objetos de automatización para herramientas programables y macro lenguajes.

Los servidores de automatización son los objetos OLE que una aplicación programable expone. Los controladores de automatización son las herramientas y programas cliente que acceden a los servidores de automatización. La librería de tipos define y registra los métodos y propiedades que un servidor de automatización expone. Permite a los clientes descubrir dinámicamente en tiempo de ejecución los servicios de automatización y sus metadatos.

[4] La carencia de herencia múltiple es considerada por la competencia como la mayor debilidad de COM.

5.3.1 Automatización y servidores de automatización

La automatización OLE, permite a un programa individual controlar a los servidores de automatización que residen en muchas aplicaciones. Permite a los integradores de sistemas ensamblar las mejores piezas de diferentes aplicaciones para crear nuevas aplicaciones. La automatización OLE permite a los clientes enlazar en tiempo de ejecución los métodos de un objeto.

El lenguaje de descripción de objetos ODL de OLE es usado para describir las interfaces que residirán en la librería de tipos de OLE.

Un servidor de automatización debe soportar la interfaz *IDispatch*. Los objetos del servidor de automatización son descritos y mantenidos en un repositorio de tiempo de ejecución llamado librería de tipos. Los programadores OLE deben describir sus objetos de automatización usando representaciones ODL de OLE. Finalmente se debe usar la utilidad *MKTYPLIB* de OLE para compilar el fichero ODL hacia la librería de tipos.

Los controladores de automatización, es decir clientes, pueden aprender los nombres de los métodos y propiedades que un objeto soporta usando la interfaz *ITypeLib* de la librería de tipos.

Un servidor de automatización OLE es un objeto OLE que implementa una interfaz específica llamada *IDispatch*. Esta interfaz provee el mecanismo de enlace retardado a través del cual los objetos pueden exponer sus funciones y datos, incluyendo sus métodos de entrada y salida así como sus propiedades, para que puedan ser invocados en tiempo de ejecución.

Un objeto proporciona funciones miembro únicas que permiten a los clientes manipular sus contenidos. Un objeto puede exponer estas funciones miembro en la forma de métodos y propiedades.

Cada aplicación automatizada debe tener en la cima de todo una clase *Application*. Esta clase puede tener clases subordinadas como *Document* y *Font*. Adicionalmente, un objeto *Application* es inicializado como el objeto activo cuando la aplicación comienza. Es posible registrar explícitamente un objeto activo. Esto permite a los clientes de OLE recuperar un objeto que ya este ejecutándose en vez de crear una nueva instancia del mismo.

Del objeto de la aplicación se puede navegar en objetos incrustados usando las propiedades como punteros. De igual forma OLE define colecciones de objetos cuya característica principal es la habilidad de iterar sobre los elementos que contiene.

Las funciones que permiten el enlace retardado en OLE son llamadas interfaces de despacho. Éstas son el equivalente a las interfaces de esqueletos dinámicos de CORBA. El servidor de OLE decide en tiempo de ejecución que método es invocado en base a un identificador (el dispID) que lee del mensaje de entrada. En contraste, las interfaces

regulares de OLE exportan sus funciones a través de punteros en una tabla *vtable* que son compilados en el código fuente del cliente.

Los clientes para invocar un método de una interfaz de despacho particular llaman a una función individual *IDispatch:Invoke* y le pasan un identificador de despacho (dispID) que define de forma única a un método o a una propiedad.

IDispatch provee mecanismos para acceder y recuperar información a cerca de los métodos y propiedades de los objetos así como del método *Invoke.*

5.3.2 Creación y gestión de información de tipos, Librerías de tipos.

Las librerías de tipos de OLE son repositorios de metadatos persistentes que describe cada método de entrada del objeto, métodos de salida, y las propiedades de lectura y escritura que éste expone. Esta información es vital para los servidores de automatización y clientes (y para los controles personalizable OLE). La interfaz *IDispatch* del servidor usa la librería de tipos para obtener la descripción de las interfaces que provee a sus clientes.

El lenguaje de descripción de objetos de OLE (ODL) es usado para definir las interfaces que un objeto soporta, incluyendo las interfaces de despacho. El ODL es un fichero de texto normal que consta de las siguientes secciones principales:

- *Library* identifica y nombra de forma única a un ODL.
- *Importlib* inserta información de otro fichero de librería de tipos (.TBL).
- *Interfaces* definen los prototipos de las funciones miembro que son parte de una tabla estática *vtable*. Las interfaces pueden también tener un GUID (también llamado IID).
- *Dispinterfaces* define los métodos dinámicos y propiedades de una interfaz de despacho. Una *dispinterface* también tiene un GUID.
- *Coclass* lista todas las interfaces de entrada y salida que soporta una clase OLE. Provee los metadatos que permite enumerar todas las interfaces de entrada y salida.

Existen dos formas de grabar información de las interfaces en una librería de tipos OLE. La forma fácil es correr el programa OLD a través de un precompilador especial llamado MKTYPLIB.EXE (para crear la librería de tipos). Las forma difícil es hacerlo uno mismo escribiendo un programa que use los servicios de creación de la librería de tipos OLE, incluyendo la función API de OLE *CreateTypeLib* y las interfaces *ICreateTypeLib* y *ICreateTypeInfo.*

Es posible llevar el fichero (.TBL) con un objeto o adjuntarlo como un recurso en un servidor de automatización EXE o DLL. Además, adicionalmente a la creación de entradas de registro para un servidor de automatización, un programa de instalación debe también crear entradas para las librerías de tipos correspondientes usando la función API de OLE *RegisterTypeLib*.

Para obtener los metadatos que requieren, los clientes de la automatización OLE deben en principio ser capaces de localizar una librería de tipos y cargar sus contenidos en memoria. Para encontrarla el cliente busca por ella en el registro y la invoca pasándole el identificador de la librería a la API de OLE *QueryPathOfRegTypeLib*.

Las interfaces *ITypeLib* e *ITypeInfo* pueden usarse para consultar los detalles de cada tipo dentro de una librería. Estas dos interfaces proveen un gran número de funciones para ayudar a los clientes a navegar a través de las librerías. Un cliente puede usar estas funciones para descubrir información acerca de nombres, identificadores de interfaces y de cualquier otra cosa que sea definida vía un ODL.

Finalmente la interfaz *IProvideClassInfo* tiene una única función miembro llamada *GetClssInfo* que regresa un *ITypeInfo* para la clase del objeto. De hecho esta interfaz es la forma más fácil de que los clientes descubran y aprendan dinámicamente las interfaces de los objetos de un servidor.

5.4 OLE: TRANSFERENCIA DE DATOS UNIFORME

OLE provee un mecanismo de transferencia de datos de intercomponentes generalizado que se puede usar en un amplio rango de situaciones y a través de una variedad de medios. Este mecanismo permite el intercambio de datos usando protocolos tales como el portapapeles, mover y soltar (*drag-and-drop*) o documentos compuestos. El intercambio de datos puede ser en el mismo documento o en una aplicación o documento diferente. La transferencia de datos puede tomar lugar sobre memoria compartida o usando ficheros de almacenamiento. Notificaciones asíncronas pueden ser enviadas a los clientes enlazados cuando cambian los datos fuentes.

La interfaz *IDataObject* encapsula los datos para ser intercambiados y provee los métodos para manipularlos.

5.4.1 El modelo de transferencia de datos de OLE

Un objeto de datos es cualquier componente OLE que implementa la interfaz *IDataObject*. Para hacer sus datos disponibles, un componente OLE inicializa un objeto de datos y crea un puntero disponible a su interfaz *IDataObject*. Un cliente puede obtener este puntero directamente o usando los protocolos de transferencia de OLE. Una vez que tiene el puntero el cliente puede invocar las funciones miembro para consultar la disponibilidad de los datos en el formato requerido, para enviar y recibir los datos actuales y para recibir las notificaciones a cerca de un cambio en los datos. Un protocolo de transferencia OLE es un mecanismo para pasar un puntero *IDataObject* de la fuente de datos a los datos del cliente.

Un objeto de datos puede representar sus contenidos de datos usando tres formatos: **estándar** que incluye los tipos de datos de intercambio del portapapeles de windows

tales como *CF_TEXT*; **privado** que son comprendidos solo por las aplicaciones que ofrecen el formato y finalmente **OLE** que define dos estructuras para el intercambio de formatos de datos:

- *FORMATETC* incluye una estructura que contiene una descripción detallada de los que esta siendo pasado.
- *STGMEDIUM* que describe el medio usado para la transferencia de datos.

Es posible descubrir que formato de datos un objeto de datos proporciona usando el llamado *Object::EnumFormatETC*.

La interfaz *IDataObject* juega un papel principal en la transferencia de datos. Provee las funciones miembro para recuperar, almacenar, enumerar datos y para gestionar notificaciones de cambio de datos.

5.4.2 Trasferencias por portapapeles, mover y soltar, y enlazado de datos

En las trasferencias por el **portapapeles** todo lo que realmente se hace es pasar un puntero *IDataObject* al portapapeles y recuperarlo cuando un cliente solicite algunos datos. OLE soporta trasferencias retardadas (*delayed),* una técnica en donde los datos no son transferidos hasta que son realmente requeridos por un cliente. Una aplicación fuente crea un objeto de transferencia de datos que mantiene una copia de los datos seleccionados y expone métodos para recuperar los datos y recibir notificaciones de cambio. Cuando un 'pegar' ocurre, la aplicación receptora hace un llamado para obtener los datos actuales de la aplicación fuente.

Mover y soltar (drag-and-drop) no es más que otro protocolo para transferir un puntero *IDataObject* entre un dato fuente y uno destino. En contraste a la transferencia retardada del portapapeles, mover y soltar es inmediato. Mover y soltar es un protocolo cooperativo entre el objeto fuente y destino. El objeto fuente genera punteros visuales que provee el usuario mediante la retroalimentación durante el proceso de mover. Finalmente, realiza cualquier acción en los datos originales como resultado de la operación soltar. El objeto destino registra su interfaz soltar. Si un soltar ocurre, el objeto destino es responsable de integrar los datos proporcionados con sus propios contenidos de datos.

Las transferencias de datos **enlazados** permiten que los clientes intercambien información al actualizar una notificación de cambio en los datos. El servicio de notificación de datos de OLE es el medio por el que los clientes se enteran de cambios en los datos fuente. Este servicio es un mecanismo de publicación y subscripción a través del cual un objeto de datos asíncronamente notifica a sus subscriptores de algún cambio en sus datos. Cuando un subscriptor (o cliente) recibe tal notificación, éste puede solicitar una copia de actualización de los datos. Para gestionar las notificaciones de datos de este tipo, OLE introduce dos interfaces: *IAdvisedSink* e *IDataAdviseHolder*.

5.5 OLE: ALMACENAMIENTO ESTRUCTURADO Y MONIKERS

OLE provee almacenamiento estructurado que permite independientemente desarrollar componentes y guardar sus contenidos dentro del mismo fichero. OLE lo llama como una *Arquitectura de Almacenamiento Estructurada.* La arquitectura de OLE provee mecanismos para que los componentes guarden su estado persistente y para que los clientes encuentren estos objetos persistentes y los carguen en memoria.

5.5.1 Almacenamiento estructurado de OLE: ficheros compuestos

OLE provee un conjunto de interfaces que provee un “sistema de archivos dentro de un fichero”. El sistema de archivos de un fichero es creado al proveer un nivel extra de indirección entre un fichero y las aplicaciones que use. Como resultado, es posible subdividir un fichero individual en compartimentos múltiples de almacenamiento que el sistema puede gestionar en función de componentes que no saben nada unos de otros.

OLE crea una jerarquía dentro de un fichero compuesto que consiste de elementos como directorios llamados *almacenes* y de elementos como ficheros llamados *flujos*. Ambos almacenes y flujos son objetos OLE. Un objeto almacén puede contener otros objetos almacén al igual que flujos. OLE permite que se creen cualquier número de almacenes y flujos dentro de un fichero compuesto individual.

Cada fichero compuesto tiene un almacén raíz del cual todos lo otros elementos descienden. Es posible usar la interfaz *IStorage* para interactuar con los objetos almacén. Esta permite mover, copiar, renombrar, crear, destruir, y enumerar elementos almacén[5].

IStorage provee dos funciones particularmente útiles: *Commit* y *Revert*. Se usa *Commit* para aceptar todos los cambios hechos en un objeto almacén desde que fue abierto por última vez. *Revert* descarta cualquier cambio hecho a este *almacén*.

Se puede acceder a cualquier elemento de almacenamiento OLE en modo directo o en modo transaccional. En modo directo, los cambios que se hacen a un objeto son inmediatos y permanentes. En modo transaccional, los cambios hechos son enviados a un buffer para que éstos puedan ser después aceptados o cancelados.

Los flujos son contenedores de datos de usuario en un fichero compuesto de OLE. Se usa la interfaz *IStream* para manipular los contenidos de los flujos y el acceso a sus datos. Los flujos dentro de un documento compuesto son muy útiles porque permiten abrir flujos como ficheros sin requerir un manejador de fichero por apertura. Sólo el objeto almacén raíz requiere un manejador de fichero.

[5] Como se puede apreciar en cada documento se encuentra un sistema de ficheros completo al estilo DOS.

5.5.2 Objetos persistentes

OLE define un conjunto de interfaces de almacenamiento persistentes que los objetos pueden implementar para guardar su estado y restablecerlo posteriormente. Un objeto persistente OLE es un objeto que puede leerse y escribirse a un almacén.

Un objeto puede soportar más de una interfaz persistente para adecuarse a diferentes contextos de almacenamiento. Un cliente debe en principio consultar a un objeto para descubrir que interfaces persistentes soporta. Las interfaces persistentes que un objeto puede implementar son:

- *IPersistStorage* permite que un objeto pueda leer y escribir su estado persistente a un objeto *IStorage*.
- *IPersistStream* permite que un objeto puede leer y escribir su estado persistente a un objeto *Stream*.
- *PersistFile* permite que un objeto pueda leer y escribir su estado persistente a un fichero en el sistema de ficheros existente.

Aunque un objeto puede soportar más de una interfaz persistente, un cliente puede solo puede usar una interfaz con una instancia dada de un objeto.

5.5.3 Monikers

Un moniker es un objeto que actúa como un nombre alias persistente para otro objeto. Los monikers pueden proporcionar sobrenombres para ficheros distribuidos, consultas de base de datos, etc. El moniker es esencialmente un nombre que identifica el origen de una pieza particular de datos, esta puede extraer los datos en una forma que es útil al cliente. Este nivel de indirección permite a los monikers transparentemente proveer los servicios.

Un moniker es simplemente un objeto persistente que implementa la interfaz *IMoniker*. Éste es usado para asignar un nombre persistente a una instancia de un objeto individual. La clase *moniker* define la funcionalidad; un objeto *moniker* mantiene los parámetros. Así que aunque todos los monikers soporten la misma interfaz, ellos tienen diferentes implementaciones las cuales son identificadas por su CLSID.

La interfaz *IMoniker* incluye la interfaz *IPersistStream*. Esto significa que los monikers pueden ser grabados en y cargados de flujos. La implementación de un moniker es identificada por su CLSID persistente. Cada clase moniker puede almacenar datos arbitrarios; ésta puede también ejecutar código arbitrario. OLE implementa cinco clases de monikers, cada una con una CLSID diferente. Estas cinco clases: *generic composite, file, item anti* y *pointer* son implementaciones polimorfas simples de la interfaz *IMoniker*. Finalmente, los clientes que trabajan con monikers usan la interfaz *IMoniker*.

Como dijimos anteriormente, puede haber muchos tipos de monikers, dependiendo de la información que ellos contienen y del tipo de objetos a los que ellos hacen referencia. COM especifica un moniker estándar llamado moniker genérico compuesto. Adicionalmente OLE implementa cuatro tipos de monikers necesarios para la unión de un documento compuesto que son:

- *Monikers de Fichero:* proporcionan una envoltura para una ruta en el sistema de archivos. El nombre del fichero puede ser absoluto o relativo.
- *Monikers de elemento:* son usados con un moniker de fichero para describir los contenidos dentro de un fichero que pueden ser tratados como elementos separados. Por ejemplo un moniker de elemento puede hacer referencia a un rango de celdas dentro de una hoja de calculo.
- *Monikers para deshacer*[6]*:* éstos son usados para negar los efectos de otro moniker.
- *Monikers compuestos genéricos:* son colecciones de otros monikers incluyendo monikers compuestos. Por ejemplo *MyDoc.doc!SalesTbl!R2C2-R4C4* selecciona los datos de la hoja de cálculo dentro de la gráfica SalesTbl que esta dentro del documento Mydoc.doc.

5.6 OLE: OCXS Y DOCUMENTOS COMPUESTOS

Un documento compuesto OLE media las interacciones entre dos tipos de componentes: contenedores y servidores. El contenedor es el componente que gestiona el documento, media el entorno visual, y gestiona las relaciones entre los servidores que representan el contenido del documento.

5.6.1 El modelo de documentos compuestos de OLE

El modelo de OLE de documentos compuestos trabaja con composición. Ambos los contenedores y los servidores son objetos OLE y existen diversos tipos de ellos. La primera clasificación esta basada en el empaquetado. Algunos servidores llamados como *en proceso (in process)*, son implementados como DLLs que comparten el mismo espacio de direcciones que el contenedor. Por su parte los servidores locales son implementados como programas separados que residen en un fichero EXE.

La siguiente división es por su función. Algunos servidores son *activos en el lugar (active in place)* otros no lo son. El enlace y el incrustado crean otro conjunto de divisiones. Los datos pueden ser incrustados o enlazados en un contenedor. Un servidor **TOTAL/(full)** soporta ambos enlazado e incrustado. Un miniservidor sólo soporta incrustado.

OLE soporta dos tipos de documentos compuestos enlazados e incrustados. Cuando un objeto es enlazado, los datos fuente residen en el mismo sitio donde fueron creados,

[6] Llamados también antimonikers

solo una referencia, o enlace al objeto y a la presentación de datos apropiada es guardada dentro del documento compuesto. El enlazado permite que los cambios hechos en el documento fuente sean automáticamente reflejados en cualquiera de los documentos compuestos que tengan el enlace. Los objetos enlazados no pueden editarse *en el lugar*. (significa que no se pueden editar sin abandonar el documento principal).

En el caso de un objeto incrustado, una copia del objeto original es físicamente almacenado en el documento compuesto por lo que el objeto llega a ser parte del documento. Es posible editar o activar objetos *en el lugar*.

Un contenedor en su mínima expresión es una aplicación OLE que puede contener objetos servidor en un documento compuesto. Un documento compuesto es típicamente un fichero compuesto OLE que contiene los datos nativos del contenedor así como objetos de varias aplicaciones servidoras. El contenedor provee un almacenamiento incrustado o un puntero de enlace persistente en la forma de un moniker para cada objeto servidor asociado con él.

Las aplicaciones servidoras crean y sirven objetos enlazados o incrustados de una clase específica. Estas aplicaciones varían en el número de objetos que pueden crear, servir y almacenar persistentemente.

La interfaz *IOleObject* gestiona la mayoría de las interacciones contenedor/servidor. Esta provee la mayoría de los métodos para la gestión de objetos en documentos compuestos.

En su máxima capacidad la combinación de contenedor/servidor debe ser capaz de realizar enlaces, edición en el lugar y envíos a memoria cache de imágenes(caching). El enlace permite a los datos de los objetos vivir en cualquier lugar, mientras que las vistas viven en el documento compuesto. La edición en el lugar permite a las vistas de objetos servidores múltiples compartir la ventana de un documento individual en sus menús. Finalmente los envíos a memoria cache permiten que un contenedor de acceso a representaciones bitmap de vistas que pueden ser desplegadas cuando en objeto no esta activo.

OLE permite incrustar los contenidos de los datos de los servidores en proceso y locales. El contenedor pasa una interfaz *IStorage* a sus servidores para permitirles gestionar el contenido de sus datos.

La otra manera de compartir un objeto es a través de un enlace. En este caso, el documento contenedor contiene un moniker que hace referencia a la localización donde los datos del objeto son almacenados. Sólo los servidores totales soportan enlazado. No es posible enlazar al servidor local o a un OCX. Observe que el servidor total es un contenedor/servidor.

5.6.2 Controles personalizados de OLE (OCXs)

Un OCX es una combinación de un servidor OLE en proceso y un servidor de automatización de OLE. En realidad es un objeto servidor en proceso que soporta

incrustación, edición en el lugar, activación de entrada y salida, automatización OLE, notificación de eventos, y objetos conectables. Adicionalmente, los OCXs soporta licencias, edición de propiedades y definen las propiedades estándar y eventos. Finalmente los OCXs se registran a sí mismos.

Los OCXs extienden la arquitectura de documentos compuestos de OLE al crear nuevas formas de iteración entre los objetos servidor OCX y sus contenedores. Ellos pueden transformar las acciones de usuarios finales, como pulsaciones del ratón en notificaciones de eventos que envían al contenedor.

Un OCX es un control bien empacado que comunica con cualquier contenedor que intente usar métodos, propiedades y eventos. Un contenedor control es un contenedor de objetos incrustados estándar.

Adicionalmente de soportar las interfaces de documentos compuestos, un OCX es también un cliente servidor de automatización OLE. Es un cliente para los servicios de automatización del contenedor, e implementa *IDispatch* para mostrar sus propias propiedades y métodos.

Un **evento** es una notificación lanzada por un OCX en respuesta a las acciones de algunos usuarios, tales como pinchar el ratón o pulsar una tecla, o cualquier acción que modifique el control. Los eventos pueden también ser emitidos en forma programada o pueden ser lanzados por el sistema.

En el modelo OCX, los eventos son lo opuesto a la invocación de métodos. El OCX expone el evento y esta muy dispuesto a llamar una implementación proveída por algún otro objeto cuando el evento se lanza e inclusive pasa los parámetros. Así que un evento es simplemente una invocación a un método de salida.

Un objeto OLE puede soportar interfaces de salida que proveen una forma estándar de definir eventos y sus parámetros. Los objetos que soportar interfaces de salida son llamados objetos conectables o fuentes.

5.7 COM DISTRIBUIDO (DCOM):

COM distribuido de Microsoft [MICR96] extiende el modelo de objetos compuestos (COM de Component Object Model) para soportar comunicación entre objetos en diferentes ordenadores en una LAN, una WAN o inclusive en Internet. Con DCOM las aplicaciones pueden ser distribuidas en las localizaciones en donde tenga más sentido para el cliente y para la aplicación.

5.7.1 La arquitectura básica de DCOM:

El modelo de objetos compuestos distribuidos DCOM es un protocolo que habilita a los componentes de software comunicarse directamente sobre una red de una forma confiable, segura y eficiente. DCOM fue diseñado para usarse a través de múltiples protocolos de red, incluyendo los protocolos de Internet tales como HTTP. DCOM está basado en la especificación DCE RPC de la fundación para software abierto(Open Software Fundation) y trabajará con los applets de Java y con los componentes ActiveX a través del uso de COM.

DCOM [BROW96] es un protocolo de nivel de aplicación para llamadas a procedimientos remotos orientados a objetos por lo que también se le llama *Object RPC*. El protocolo consiste de un conjunto de extensiones, basadas en la especificación RPC en el entorno de computación distribuida (DCE). DCOM especifica:

- Como las llamadas son hechas en un objeto.
- Como las referencias de un objeto son representadas, comunicadas y mantenidas.

COM [MICR96] define como los componentes y sus clientes interactuan. Esta interacción puede definirse como aquella en que el cliente y el componente pueden conectarse sin la necesidad de ningún componente intermediario del sistema. El cliente llama a los métodos en el componente sin ningún overhead whatsoever.

Un cliente que necesita comunicarse con un componente en otro proceso no puede llamar al componente directamente, sino que tiene que usar alguna forma de comunicación entre procesos proveída por el sistema operativo. COM proporciona esta comunicación de una forma completamente transparente: intercepta las llamadas del cliente y las envía al componente en otro proceso

Cuando el cliente y el componente residen en máquinas diferentes, DCOM simplemente reemplaza la comunicación local entre procesos con un protocolo de red. Ni el cliente ni el componente se dan cuenta que el cable que les une se ha hecho más largo.

La figura 5.2 muestra la arquitectura general de DCOM: El run-time de COM provee servicios orientados a objetos a sus clientes y componentes y usa RPC y el proveedor de seguridad para generar paquetes de red estándar que conforman el protocolo estándar DCOM.

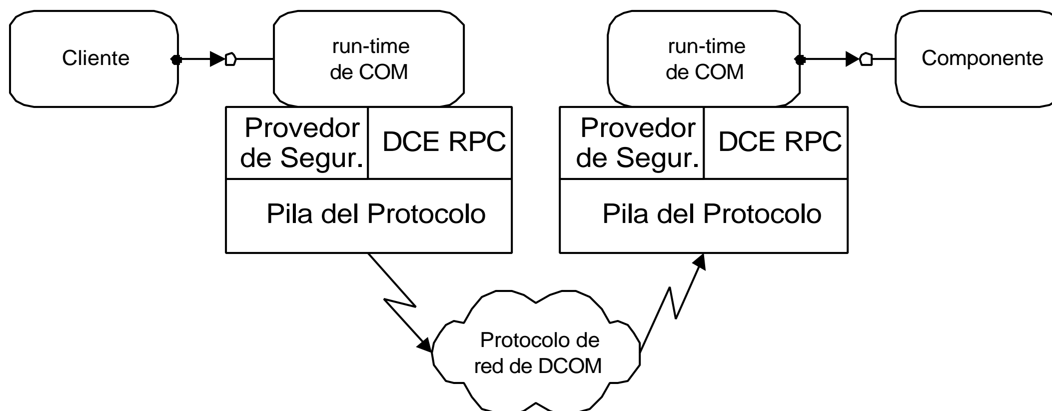


Figura 5.2 – Arquitectura General de DCOM

5.7.2 Condiciones Seguridad

DCOM [BROW96] permite la autentificación, autorización y las capacidades de integridad de mensajes de DCE RPC. Una implementación debe soportar algún nivel de seguridad DCE RPC. Cualquier conexión o llamada puede ser hecha como segura o como insegura de acuerdo a como sea negociada por el cliente y el servidor.

En DCOM se le puede dar una versión a las interfaces, esto es hecho a través de identificadores universalmente únicos (UUID´s). Para dar una versión a una interfaz publicada, la interfaz es definida con un UUID diferente para actualizar la especificación

En general, como cualquier protocolo de transferencia de datos genérico, DCOM no puede regular el contenido de los datos que son transferidos, ni existe ningún método a priori para determinar la sensibilidad de alguna pieza particular de información dentro del contexto de cualquier ORPC dado.

Sin embargo, DCOM provee completamente la infraestructura de seguridad definida por DCE RPC, la cual permite varias formas de autentificación, autorización e integridad de mensajes.

PARTE II.- LA TECNOLOGÍA DE AGENTES MÓVILES Y SUS NECESIDADES ACTUALES.

Capítulo VI
AGENTES: INTRODUCCIÓN, CLASIFICACIÓN Y CONCEPTOS BÁSICOS

Los agentes inteligentes son un objeto de estudio popular de muchas investigaciones hoy en día en campos como Psicología, Informática[7] e Inteligencia Artificial (IA) en donde se estudia más intensamente. Sin embargo aunque cada día se toman más cartas en el asunto resulta difícil contestar a la pregunta de ¿Qué es un agente?, Debido a los diversos entornos en los que este concepto se utiliza.

Así pues este capitulo tiene como finalidad analizar brevemente el origen de los agentes así como las características que éstos suelen tener en los diferentes ámbitos para posteriormente precisar el tipo de agente que utilizaremos en este trabajo.

6.1 INTRODUCCION

El estudio de agentes se remonta varios años atrás, sin embargo en principio parecía que el concepto de agente estaba enfocado a la obtención de objetivos diferentes a los que se tienen hoy en día.

La IA fue una de las áreas que empezó a trabajar en este tema, la finalidad fundamental era crear un lenguaje y protocolo para que los agentes inteligentes lograran compartir conocimiento. Los agentes en este ámbito, era común que poseyeran conceptos que son usualmente aplicados a humanos, ya que se pretendía que los agentes imitaran nuestro comportamiento.

De esta forma la Agencia de proyectos de investigación avanzada ARPA [UBMCa] (Advanced research project agency) financió el *esfuerzo de compartir conocimiento* (Knowledge Sharing Effort) que esta desarrollando metodologías y software para la compartir y reusar conocimiento. Mucho de esto es muy relevante para la construcción

[7] Aunque en este capitulo mencionares conceptos referentes a inteligencia artificial el estudio de los agentes que haremos en este trabajo será referido al ámbito de agentes móviles e Internet.

de sistemas basados en agentes tales como el lenguaje de manipulación y consultas de conocimiento KQML (Knowledge Query and Manipulation Language). KQML es un lenguaje y protocolo para el intercambio e información de conocimiento cuya finalidad es proporcionar un lenguaje común para la comunicación de agentes inteligentes.

Mientras esto sucedía en IA, por otro lado General Magic [LAWT96] era fundada en 1990 para entregar una tecnología basada en agentes para dispositivos de computo portables. Su tecnología Telescrip fue creado por AT&T, Motorola y algunas otras.

La idea era que los clientes una vez conectados a una red enviaran sus agentes a navegar para buscar noticias y gangas, después mas tarde deberían conectarse nuevamente para verificar su progreso y obtener los resultados. Desgraciadamente, Telescript nunca tuvo éxito por lo que más tarde cambiaría de giro su tecnología.

Mas tarde como veremos en evolución, el concepto y tecnología de agentes poco a poco fueron tomando una gran importancia en el ámbito informático al añadir movilidad a los agentes y darles capacidad de navegar en Internet.

Bajo este contexto es fácil mencionar que el termino "agente" ha sido sobrecargado por distintos grupos por lo que hoy en día es difícil precisar lo que un agente debe hacer. Debido a este hecho también llega a ser complicado para los usuarios hacer una buena estimación de cuales son las posibilidades de la tecnología de agentes. Hoy en día parecen existir más definiciones de este concepto que sistemas de agentes funcionando.

Merece la pena mencionar que actualmente los dos campos de estudio fundamentales de los agentes siguen siendo la inteligencia artificial e Internet en donde en los últimos años, ha tomado una fuerza impresionante su estudio; al introducirse nuevos conceptos como movilidad o nuevas capacidades de razonamiento.

De esta forma la descripción de las capacidades de los agentes no es nada fácil. Ya que en principio dependerá del área que estemos abordando y segundo quizá llegar a precisar el alcance de nuestro proyecto para poder definir a un agente.

Sin embargo existen ciertos puntos que se pueden criticar sobre todo en el campo informático en donde es muy frecuente escuchar que ellos no son una nueva técnica, y que cualquier cosa puede ser hecha con agentes, cosa que como veremos después resulta falsa. Así pues podemos criticar:

- Como los temas de estudio principales de inteligencia Artificial (sistemas expertos y redes neuronales) no han tenido el éxito esperado, ahora al añadir el concepto de agentes la IA busca una nueva manera de recuperar importancia.
- Cualquier cosa a la que se le pone la etiqueta de agente o de inteligente vende. Ahora bien se hacen llamar inteligentes al observar que al crear el sistema de agentes el código es pequeño debido a que normalmente se apoya en una arquitectura subyacente.

De esta manera llegar a la definición precisa del termino resulta una tarea ardua, por lo que intentaremos dar una definición en base a las características que un agente debe tener.

6.2 CONCEPTOS BASICOS

Antes de continuar con el estudio de esta tecnología es necesario precisar ciertos conceptos que utilizaremos mas tarde. Dada la multiplicidad de roles que un agente puede desempeñar resulta difícil y quizá poco práctico dar una sola definición, por lo que mencionaré un par de definiciones generales y posteriormente precisaré algunas características para finalmente hablar de otros conceptos.

6.2.1 Definición de agente

La primera definición más general dada por G.W. Lecky-Thompson [UBMCb] es: '*Un agente es una pieza de software que ejecuta una tarea dada usando información obtenida de su entorno para actuar de forma apropiada para completar la tarea exitosamente. El software debe ser capaz de adaptarse a sí mismo en base a los cambios que ocurren en su entorno, para que un cambio en circunstancias le permita aún obtener el resultado deseado".*

Un segundo intento más técnico orientado a la tecnología de objetos y agentes móviles es el dado por Fritz Hohl [HOHL96]: "*Un agente móvil es un objeto especial que tiene un estado de datos(otros objetos no agentes, estructuras y bases de datos), un estado de código (las clases del agente y otras referencias a objetos) y un estado de ejecución (el control de procesos que se ejecutan en el agente)"*

Como ninguna de estas definiciones resulta completa en vez de dar una definición formal, proporcionaré la lista de características que se espera que un agente deba tener, para poder tener una idea de lo que un agente puede ser. Las características siguientes suelen tenerlas los agentes móviles que serán el tipo de agentes a las que esta dedicado este trabajo.

La forma más general en que es usado el termino agente es para denotar un sistema de software basado en agentes, en este ámbito un agente debe tener las siguientes propiedades:

- *Autonomía:* un agente opera [WOOL95] sin la intervención directa de humanos y debe tener una cierta clase de control sobre sus acciones y su estado interno.
- *Habilidad social:* los agentes interaccionan con otros agentes y (posiblemente) con humanos.
- *Reactividad:* los agentes perciben su entorno y responden en un tiempo razonable a los cambios que ocurren en él. El agente puede estar en estado pasivo la mayor parte del tiempo y despertar al momento de que detecte ciertos cambios[8].
- *Proactividad:* los agentes no solo responden a cambios sino que pueden tener un comportamiento con una iniciativa propia hacia una meta dirigida.
- *Continuidad temporal:* los agentes están constantemente ejecutando procesos ya sea en forma activa o pasiva.

[8] Esta característica es muy similar a la que tienen los demonios en el entorno UNIX.

- *Orientación hacia el objetivo final:* el agente es capaz de desarrollar una tarea compleja. Para lograrla es necesario subdividir esta tarea en pequeñas subtareas, el agente debe decidir por si mismo la mejor manera y orden de ejecutarlas para lograr el objetivo final.
- *Movilidad*[9]*:* el agente debe ser capaz de suspender su ejecución en un servidor y reanudarla en otro servidor una vez que se haya desplazado a este. Este concepto se ha introducido en los últimos años.

Para algunos investigadores, especialmente para los que trabajan en IA, el termino agente tiene un significado más fuerte y específico. Estos investigadores normalmente pretenden que un agente sea como un sistema informático que además de tener las características mencionadas previamente posean conceptos aplicados normalmente a los humanos, así pues podremos añadir nuevas características a los agentes para el ámbito de IA:

- *Benevolencia:* es la suposición de que los agentes no tienen metas conflictivas y que cada agente por consiguiente tratara de hacer la tarea que le fue encomendada.
- *Racionalidad:* es la suposición de que el agente siempre actuara para lograr sus metas y de que no actuara de una forma que evite la consecución de las mismas.
- *Adaptabilidad:* un agente debe ser capaz de ajustarse a sí mismo a los hábitos, métodos de trabajo y preferencias de su usuario.
- *Colaboración*[10]*:* un agente debe tomar en cuenta que los usuarios humanos cometen errores, por ejemplo pueden omitir información o proporcionarla de una forma ambigua. Un agente debe ser capaz inclusive de negarse a ejecutar ciertas tareas si éstas implican una sobrecarga para los recursos de la red o si ocasionase daño a los usuarios.

Aunque aún no hay un solo agente que posea todas estas habilidades, existen sistemas de agentes prototipo que poseen muchas de ellas. Además nos proporcionan una buena idea de lo que un agente puede hacer, así como que características debemos de evaluar al examinar la calidad de un sistema de agentes en específico.

6.2.2 Albedrío, inteligencia y movilidad.

El grado de albedrío (de la lengua inglesa *agency*), inteligencia y movilidad que puedan tener los agentes de un sistema son los factores principales a medir para evaluar la calidad de un sistema de agentes, estas características son las que fundamentalmente les proporcionarán a los agentes un mayor grado de versatilidad para la consecución de cualquier tarea.

[9] Esta propiedad inicialmente se adjudicaba a los agentes en el ámbito de IA, sin embargo en los últimos años a sido el factor principal para el desarrollo de los agentes en el ámbito de Internet y ha llegado aser considerada una característica fundamental

[10] Este concepto fue analizado en la conferencia de agentes éticos del Web en las *memorias de la segunda conferencia internacional de WWW*, Chicago, U.S.A.

El grado de autonomía y autoridad que le es otorgado a un agente es llamado su albedrío. Éste puede ser medido al menos cualitativamente por la naturaleza de la interacción entre los agentes y otras entidades en el sistema que opera.

Como mínimo, un agente debe correr de forma asíncrona. Un agente más avanzado puede interaccionar con otras entidades tales como datos, aplicaciones, o servicios. Los agentes aún más avanzados colaboran y negocian con otros agentes.

Es difícil precisar lo que hace a un agente inteligente pero una buena definición en dada en [IBM95]: *“La inteligencia es el grado de razonamiento y comportamiento aprendido, la habilidad del agente para aceptar las premisas de los objetivos del usuario y de llevarlas a cabo las tareas que le son delegadas.*
Niveles mas altos de inteligencia incluyen un modelo de usuario o alguna otra forma de comprensión y razonamiento a cerca de lo que el usuario quiere hacer, y planear los medios para conseguir ese objetivo.
Más allá en el la escala de la inteligencia están los sistemas que aprenden y se adaptan a su entorno ambos en términos de los objetivos del usuario, y en términos de los recursos disponibles en el agente. Tal sistema podría, como un asistente humano, descubrir nuevas relaciones, conexiones, o conceptos independientemente del usuario humano, y explotarlos anticipando y satisfaciendo las necesidades del usuario."

Finalmente la movilidad es la facultad que tiene un agente para suspender su ejecución momentáneamente, trasladarse a otro servidor, volver a activarse y continuar su ejecución justo en donde la había detenido anteriormente. Al trasladarse debe llevar consigo sus datos, código y estado de ejecución para poder reanudar o bien continuar la tarea que estaba desempeñando previamente. De igual forma es obvio suponer que todas las acciones para conseguir su traslado deben ser ejecutadas sin intervención humana.

Resulta oportuno mencionar que hoy en día los sistemas de agentes móviles más populares son los basados en Java[11], estos sistemas son capaces de transportar el estado de datos, es decir sus bases de datos y estructuras; de igual forma puede transportar su estado de código, como lo son las clases y objetos que requiere, pero no su estado de ejecución que se refiere al punto exacto que se estaba ejecutando al detener la ejecución del agente, esto es debido a que la maquina virtual de Java no permite acceder a estos recursos.

6.3 CLASIFICACIÓN DE AGENTES:

Los agentes pueden ser clasificados en una variedad de formas: por sus características, por la tecnología que los soporta, por la función que desarrollan (o lo que hacen), o por una mezcla de ellos. La clasificación de agentes no es única, así como algunas de las características que de los agentes usadas para su clasificación no son mutuamente exclusivas.

[11] Entre los que podemos citar Aglets Workbench, Voyager, Mole, Java-to-go, etc.

6.3.1 Los agentes en base a su movilidad.

Basados en su movilidad los agentes pueden ser clasificados como:

- *Agentes estacionarios o estáticos:* un agente estacionario reside en un sistema informático y ejecuta sus funciones y/o tareas, interacciona con el mundo exterior intercambiando mensajes. De forma parecida al software convencional.
- *Agentes móviles:* son aquellos que ejecutan las tareas designadas en distintos ordenadores moviéndose de uno a otro para recopilar la información requerida y posteriormente regresar a su sitio de origen.

6.3.2 Los agentes en cuanto a su característica fundamental.

Como habíamos mencionado anteriormente las características fundamentales que hacen valer más o menos a un agente son: autonomía, inteligencia y movilidad. Estas tres características dan origen a otra clasificación que se ha hecho sobre los agentes que es:

- *Agentes inteligentes:* estos están diseñados para razonar interactivamente la información que se les va proveyendo para seleccionar la mejor opción. Son capaces de realizar tareas complicadas y de aprender de su entorno.
- *Agentes autónomos:* aquellos que son capaces de actuar sin intervención humana y que son capaces de tomar decisiones importantes por sí solos. Desde el punto de vista programacional éstos agentes poseen un hilo (thread) propio.
- *Agentes móviles*[12]*:* aquellos que son capaces de trasladarse de un servidor a otro para reanudar su ejecución en un nuevo sitio destino.

Como se puede apreciar las clasificaciones tienen una gran relación, es decir los agentes autónomos normalmente para lograr un buen grado de autonomía tienen que ser inteligentes. A su vez los agentes inteligentes la mayoría de las veces son autónomos aunque habrá ocasiones en que su inteligencia sólo la denoten al procesar la información que reciben de un usuario interactivamente.

Por último los agentes móviles suelen ser autónomos y en función de su grado de inteligencia esta la complejidad de las tareas que pueden desempeñar. Sin embargo hay muchos agentes móviles que realizan tareas rutinarias como la recopilación de datos en donde su grado de inteligencia en casi nulo.

6.3.3 Diversos tipos de agentes

Debido a la sobrecarga del termino que ha sido usada para todo, podremos citar diversos tipos de agentes[MURU98] que mezclan características mencionadas en las otras clasificaciones:

[12] A partir del siguiente capitulo mi trabajo se enfocará al estudio de agentes móviles.

- *Agentes de interfaz:* éstos asisten a los usuarios y proveen interfaces de usuario "inteligentes" que pueden detectar cuando un usuario esta teniendo dificultades y asesoran al usuario acerca del problema. Observan y aprenden las preferencias y hábitos del usuario para automatizar acciones que son ejecutadas cotidianamente y sugieren cursos de acción. Finalmente reducen la complejidad de la interfaz favoreciendo una mayor funcionalidad.
- *Agentes colaboradores y sistemas multiagente:* un agente colaborador interacciona y coopera con otros agentes en nombre de un usuario para ejecutar una tarea. Un sistema multiagente consiste en dos o más agentes semi-autonómos que interaccionan, colaboran y trabajan juntos para ejecutar un conjunto de tareas o lograr un grupo de objetivos.
- *Agentes de Internet/Información:* los agentes de Internet pueden filtrar una gran cantidad de información disponible en la red, pasándole al usuario sólo aquella información en la que está interesado. Ellos automatizan la recuperación y el proceso de la información obtenida de la red.
- *Agentes de aprendizaje:* su principal función es asistir a los usuarios una vez que hayan aprendido de observar e imitar sus acciones y de la retroalimentación correctiva que reciben por parte de los usuarios.

6.4 EVOLUCIÓN DE LOS AGENTES:

Debido a la gran cantidad de agentes que existen su evolución ha sido muy variada ya que han tomado giros distintos en función del ámbito en el que se desarrollan. Definitivamente como lo mencionamos en la introducción el estudio de agentes comenzó en aplicaciones de inteligencia artificial, sin embargo con la explosión del Web e Internet han nacido nuevas vertientes y tendencias por lo que hoy en día los agentes gozan de una gran relación con Internet.

6.4.1 Diversas etapas

En sus principios una de las primeras compañías que invirtió en la tecnología de agentes fue General Magic [LAWT96] que fue fundada en 1990 para crear una tecnología basada en agentes para dispositivos de computación portables. Esta tecnología llamada Telescrip fue desarrollada por AT&T, Motorola y algunos otros.

Posteriormente adaptaron Telescrip para trabajar con servidores en Internet y crearon *Tabriz AgentWare* el cual ejecutaba y gestionaba aplicaciones basadas en agentes que estaban en servidores.

En el campo de inteligencia artificial el estudio de agentes ha seguido avanzando, sobre todo al tratar de crear nuevos agentes que se asemejen más cada día al comportamiento humano y con un alto nivel de raciocinio e inteligencia que son las dos características fundamentales de estos agentes. Sin embargo al paso del tiempo con la explosión de Internet el desarrollo de estos agentes ha sido menguado por la rápida evolución de los agentes de Internet y agentes móviles.

Mientras *Telescrip* era precursora en el campo de agentes móviles se creaba el *Lenguaje de manipulación y consultas de conocimiento (KQML)*[13] que hoy en día sigue vigente. De igual forma se implementaba el lenguaje de *Agent TCL* en Dartmouth College; *Agents for remote action* (ARA) en la universidad de Kaiserslautern, Alemania y *Penguin*.

Todos estos sistemas que son sistemas de agentes móviles, salvo KQML, siguen vigentes hoy en día y cada vez se realizan nuevas versiones, sin embargo la carencia de éxito en algunos de ellos radica en que son basados en lenguajes propietarios o poco difundidos en la mayor parte de las veces, en donde para poder realizar una aplicación basada en agentes móviles es necesario aprender el propio lenguaje de codificación lo cual no resulta una tarea fácil. Peor aún por ser propietarios una aplicación hecha en telescript no podía interaccionar con otra hecha en ARA por ejemplo. Debido a esto el avance no fue significativo.

En 1995 con la aparición de windows '95 el mundo de las interfaces gráficas dominó el mercado completamente y dio lugar a la proliferación de los agentes de interfaces, de hecho Microsoft ofrece un producto llamado Microsoft Agent.

De igual forma la explosión del número de usuarios en Internet sufrió un gran incremento de 1995 a 1996, ya que el número de servidores fue prácticamente duplicado por lo que los agentes de Internet que ya existían tomaban más fuerza.

De igual forma el tráfico excesivo en la red así como el sueño de crear una tecnología que realizará las tareas tediosas en Internet hizo buscar a los investigadores una nueva forma de solventar el problema por lo que se retomó el tema de agentes móviles. Aunado a esto se presenta el gran éxito del lenguaje Java, cuya mayor ventaja era la independencia de la plataforma y ofrecía una base segura para el desarrollo de agentes móviles.

Java se encontraba difundido en todo el mundo y brindaba características fundamentales como la invocación de métodos remotos (RMI) y la serialización para implementar sistemas de agentes móviles. Quizá debido a esto ha sido la plataforma que ha tomado más fuerza para el desarrollo de agentes móviles y de la que se espera ver lo mejor en los próximos años.

6.4.2 De agentes inteligentes a agentes móviles

Poco a poco se ha ido transformando la importancia del atributo *inteligencia* por el de *movilidad*, ciertamente se ha analizado que crear agentes muy inteligentes suele ser demasiado costoso y en ocasiones poco fructífero, por otro lado crear agentes móviles es relativamente sencillo y para ciertas aplicaciones brindan grandes beneficios y mejoran en el desempeño de la aplicación.

[13] El cual no es un sistema de agentes móviles sino para compartir conocimiento.

La necesidad de una nueva tecnología que ayudará a resolver el tráfico en Internet así como lograr la independencia de tareas rutinarias en Internet ha hecho que en los últimos años haya habido una gran proliferación de sistemas de agentes móviles.

El concepto de movilidad cada día ha sido más aceptado y valorado al empezarse a comprender los beneficios de la tecnología, de igual manera otras tendencias como la venta de productos por Internet a favorecido la creación de diversos sistemas de software en donde no sólo se exige que los agentes deban ser móviles sino inteligentes y autónomos también.

A pesar de que los nuevos sistemas exijan todas las características deseables en un sistema de agentes, la *movilidad* cada día juega un papel más relevante por lo que muchas aplicaciones para el desarrollo de agentes crean nuevas versiones incluyendo movilidad o bien se desarrollan nuevas, entre ellas podríamos citar una larga lista de aplicaciones de agentes móviles que se encuentran hoy disponibles en el mercado.

La necesidad de una nueva tecnología que ayudará a resolver el tráfico en Internet así como lograr la independencia de tareas rutinarias en Internet ha hecho que en los últimos años haya habido una gran proliferación de sistemas de agentes móviles. Por ejemplo:

- Agent Tcl (Dartmouth College)
- Aglets Workbench (IBM)
- ARA - Agents for Remote Action (University of Kaiserslautern)
- Concordia (Mitsubishi Electric Information Technology Center America)
- ffMAIN - The Frankfurt Mobile Agents Infrastructure (Johann Wolfgang Goethe Universität)
- Java Agent Template (H. Robert Frost)
- JATLite (Stanford University)
- MOA - Mobile Objects and Agents (OpenGroup)
- Mole (Mole Team)
- Odyssey (General Magic)
- Voyager (ObjectSpace)

La mayor parte de estos sistemas de agentes móviles o frameworks están implementados en Java, no obstante prevalecen algunos que siguen siendo en lenguajes propietarios, analizaremos los más importantes en posteriores capítulos.

6.5 AGENTES MÓVILES

Es importante antes de profundizar en la tecnología que se de una definición detallada de agente móvil por lo que presentaré mi definición personal:

Un agente móvil es un programa con características especiales que presenta tres estado; un estado de datos(estructura de datos y otros objetos no agentes), un estado de código

(las clases del agente y otras referencias a objetos) y un estado de ejecución (el control de procesos que se ejecutan en el agente).

Accede a un conjunto de métodos que le permiten moverse de un servidor a otro, siendo ésta su característica principal. El agente posee un programa de ejecución principal que define las tareas que realizará y el grado de inteligencia con el que actuará para interactuar con los usuarios y para resolver tareas adversas en su entorno. Finalmente es autónomo porque no necesita consultar a ningún supervisor humano para tomar una decisión y porque posee un hilo de ejecución propio e independiente del resto de los agentes.

Las características básicas de un agente móvil son:

- *Autonomía*: tienen su propio hilo y actúan por sí mismos.
- *Móvil:* serializable.
- *Concurrente:* puede ejecutarse con mas agentes a la vez.
- *Direccionable*: se puede definir su comportamiento.
- *Continuo*: se ejecutan continuamente y por tiempo indefinido.
- *Reactivo*: reacciona a su entorno mediante métodos.
- *Social*: interoperan con objetos y otros agentes.
- *Adaptativo:* lo gestiona con excepciones.

6.5.1 El paradigma de Agentes móviles

Debido a la dificultad de la arquitectura actual de Internet para satisfacer el ritmo de crecimiento exponencial de sus los usuarios, era necesario una nueva aproximación que satisfaciera dos necesidades aparentemente contradictorias: incrementar la sofisticación de los tipos posibles de comunicación sin restringir el ancho de banda disponible de los componentes de Internet. Una solución que satisface ambas necesidades son los agentes móviles[WHIT96].

6.5.1.1 Aproximación actual

El principio de organización central de las comunicaciones en las redes de ordenadores de hoy en día, que son las llamadas procedimientos remotos RPC (de la lengua inglesa *Remoto Procedure Call)*, los cuales habilitan a un ordenador llamar procedimientos en otro (Véase figura 6.5.1.1).

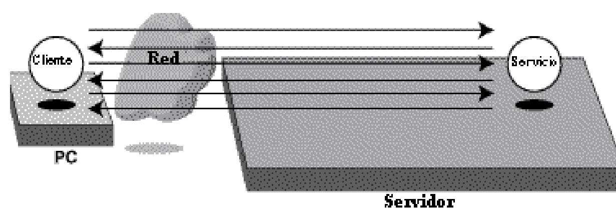


Figura 6.5.1.1 - Aproximación mediante el uso de RPC

Cada mensaje en la red transporta solicitudes o confirmaciones de procedimientos. Una solicitud incluye datos que son los argumentos del procedimiento y la respuesta incluye datos que son sus resultados. El procedimiento mismo es interno al ordenador que lo ejecuta. Y todos los mensajes viajan desde el punto origen al destino

6.5.1.2 Nueva aproximación

Una alternativa a las llamadas de procedimientos remotos es la promoción remota (obsérvese la figura 6.5.1.2). Dos servidores que se comunican con el paradigma de programación remota hacen un acuerdo sobre las instrucciones que son permitidas en un procedimiento y los tipos de datos que son permitidos en su estado. Estos acuerdos constituyen un lenguaje. El lenguaje incluye instrucciones que permiten al procedimiento tomar decisiones, examinar, y modificar su estrado, y llamar procedimientos proporcionados por el ordenador que se está recibiendo el programa. Tales llamadas a procedimientos son locales en vez de remotas. El procedimiento y su estado son llamados **agente móvi**l para enfatizar que ellos representan al ordenador que los envía aunque ellos residen y operan en el ordenador que les recibe.

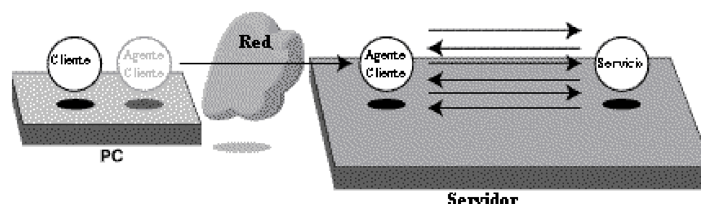


Figura 6.5.1.2 - Nueva aproximación mediante programación móvil

6.5.1.3 Ventajas de la programación móvil

La ventaja táctica inmediata de la programación remota es el desempeño. Ya que será más rápido enviar un agente al servidor y directamente trabajar localmente en vez que remotamente.

Por otra parte la ventaja estratégica de la programación remota es la personalización. La programación remota cambia no sólo la división de tareas entre los fabricantes de software, sino también facilita la instalación de ese software. Los componentes del servidor de una aplicación basada en programación remota, son dinámicamente instalados por la misma aplicación que se está ejecutando desde el ordenador del usuario, desde que cada componente es un agente.

6.5.2 Conceptos de Agentes móviles

La primera implementación comercial del concepto de agentes móviles fue la tecnología de Telescript de General Magic que intentó permitir acceso automático e interactivo a las redes de ordenadores usando agentes móviles.

Telescript fundó las bases de la tecnología de agentes móviles y es reconocido precursor de la misma. Estos conceptos fueron tan bien diseñados que han sido utilizados por los nuevos sistemas de agentes de nuestros días.

6.5.2.1 Lugares

La tecnología de agentes modela una red de ordenadores, tan grande como una colección de lugares que ofrecen un servicio a los agentes móviles que entran en él. Así pues un lugar es un sitio de interacción de agentes que les ofrece un servicio y les facilita la consecución de sus atareas a todos aquellos agentes a los que les haya sido permitido entrar. En la tecnología de telescript un lugar era ocupado permanentemente por un agente distinguido. Este agente estacionario representaba al lugar y proveía su servicio (Obsérvese la figura 6.5.2.1).

6.5.2.2 Agentes

Cada agente móvil que fue definido anteriormente, ocupa un lugar particular. Sin embargo un agente puede moverse de un lugar a otro, de esta forma ocupa lugares diferentes en tiempos diferentes. Los agentes son independientes por lo que sus procedimientos son ejecutados concurrentemente.

Figura 6.5.2.1 - Representación de lugares y agentes

6.5.2.3 Viajes

A los agentes les es permitido viajar de un lugar a otro, sin importar la distancia, esto es el sello distintivo de un sistema de programación remota. De esta forma un viaje le permite a un agente obtener un servicio ofrecido remotamente y regresar a su lugar de origen.

6.5.2.4 Entrevistas (meetings)

A dos agentes les es permitido entrevistarse si ellos se encuentran en el mismo lugar. Una entrevista les permite a los agentes en el mismo ordenador llamar los procedimientos de otros.

Figura 6.5.2.2 - Viajes y entrevistas

Las entrevistas motivan a los agentes a viajar. Un agente podría viajar a un lugar en un servidor para entrevistarse con el agente estacionario que provee el servicio que el lugar ofrece. La Figura 6.5.2.2 describe las interacciones en los viajes y entrevistas.

6.5.2.5 Conexiones

Una conexión de comunicación entre dos agentes en lugares distintos, es con frecuencia hecha en beneficio de los usuarios humanos para que interactuen con las aplicaciones. La instrucción de conexión permite a los agentes de los usuarios el intercambio de la información a distancia. Por ejemplo un agente podría enviar información al sitio de donde proviene (a través de otro agente) para que el usuario origen pueda seleccionar alguna opción de la información que ha encontrado.

6.5.2.6 Autoridades

La autoridad de un agente o lugar en el mundo electrónico es el individuo, organización entidad o empresa a quien representa dentro del mundo físico. Una autoridad puede ser propietaria de varios agentes, y autenticación de los agentes generalmente consiste en descubrir su autoridad. Los agentes y los lugares pueden discernir, pero nunca podrán negar ni falsificar sus autoridades, evitando el anonimato.

6.5.2.6 Permisos

Las autoridades pueden limitar los que los agentes y lugares puede hacer asignándoles permisos a ellos. Un permiso es un dato que acepta capacidades. Un agente o lugar puede darse cuenta de sus capacidades, lo que le es permitido hacer, pero nunca podrá incrementarlas.

6.5.3 La tecnología de agentes móviles

Una tecnología para agentes móviles es una infraestructura de software que puede montarse encima de una gran variedad de computadoras y hardware de comunicaciones, presente y futuro. Esta tecnología, implementa los conceptos mencionados en la sección anterior y otros relacionados con ellos que les permite a los agentes interoperar.

La tecnología tiene tres componentes principales: el lenguaje en el cual los agentes y los lugares son programados; una máquina o interprete para ese lenguaje; y los protocolos

de comunicación que permite a esa máquinas residir en diferentes ordenadores para lograr el envío en intercambio de agentes.

6.5.3.1 El lenguaje

El lenguaje de programación de los creadores de aplicaciones de comunicación con agentes debe definir los algoritmos que los siguen los agentes y la información que llevan conforme viajan por Internet.

Para facilitar el desarrollo de las aplicaciones de comunicación, y la interacción entre lugares y agentes el lenguaje de programación debe ser:

- *Completo:* para que cualquier algoritmo pueda ser expresado en el lenguaje.
- *Orientado a objetos:* para obtener los beneficios de esta tecnología
- *Dinámico:* para que pueda transportar la clases que se requieran para crear instancias de agentes en máquinas remotas.
- *Persistente:* para que el agente y su información sean respaldados en un medio no volátil.
- *Portable y seguro*: para que se pueda ejecutar sobre cualquier plataforma de una forma segura.
- *Versátil:* que permita tareas complejas de transportación, autenticación y control de acceso.

6.5.3.2 La máquina o interprete

La máquina es un programa de software que implementa el lenguaje para mantener y ejecutar los lugares dentro de su contexto, al igual que a los agentes que ocupan esos lugares.

Al menos conceptualmente la máquina consta de tres interfaces de programación de aplicaciones (APIs), obsérvese la figura 6.5.3.1. Una API de almacenamiento que le da acceso a memoria no volátil que requiere para preservar a los lugares y agentes en caso de fallas. Una API de transporte que le da a la máquina acceso a los medios de comunicación que requiere para transportar agentes a y de otras máquinas. Y una API de aplicaciones externas que le permite a las partes escritas en el lenguaje de agentes interactuar con otras que no están escritas en dicho lenguaje.

6.5.3.3 Protocolos

Los protocolos habilitan a dos máquinas que se comuniquen. Las máquinas se comunican para transportar agentes. Los protocolos deben operar sobre una gran cantidad de redes de transporte incluyendo aquellas basadas en el protocolo de Internet TCP/IP.

Los protocolos operan en dos niveles. El nivel más inferior gestiona el transporte de los agentes; y el más alto, su codificación y decodificación.

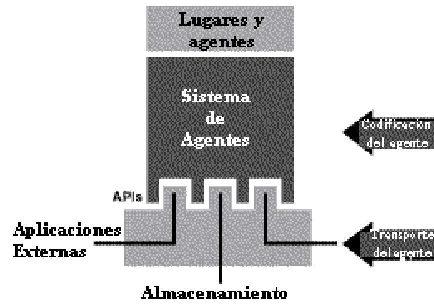


Figura 6.5.3.1 - La máquina y los niveles de los protocolos

El protocolo de interconexión de plataformas especifica como dos máquinas se autentifican una a otra usando por ejemplo, criptografía de llave pública y entonces transfiere la codificación de un agente de una a otra.

Capítulo VII

EL POR QUÉ Y EL PARA QUÉ DE LOS AGENTES MÓVILES.

En el capítulo anterior mostramos los conceptos básicos de la tecnología de agentes móviles así como la evolución que han tenido en los últimos tiempos. En este capítulo veremos brevemente la diferencia de los agentes móviles con otras tecnologías de programación distribuida, los beneficios que ofrece esta tecnología que le han hecho tomar tanta fuerza, así como las aplicaciones que se pueden desarrollar con ella.

7.1 LOS AGENTES MÓVILES Y OTRAS TECNOLOGÍAS

La tecnología de agentes móviles es una tecnología emergente que se han generado de la recopilación de las mejores características de otras tecnologías de programación distribuida. De aquí, que los agentes móviles son el resultado de objetos distribuidos, software intermedio orientado mensajes, applets, etcétera.

A continuación mencionaré brevemente algunas diferencias de los agentes móviles con algunas otras tecnologías de programación distribuida.

7.1.1 Objetos distribuidos, software intermedio[14] orientado a mensajes y agentes móviles

Hoy en día, la mayoría de los programadores de Java están familiarizados con los objetos distribuidos [SOMM97] y cómo éstos pueden ser usados para dividir un sistema a través de varias máquinas dentro de una red, como lo muestra la figura 7.1.1. La invocación de métodos remotos(RMI) de Java y varias implementaciones de la especificación de CORBA están disponibles para los programadores de Java. Usando

[14]de la lengua inglesa middleware

objetos distribuidos, el programador de Java puede invocar métodos de objetos que residen en otras máquinas tan fácilmente como invocar métodos en objetos locales.

Adicionalmente a los objetos distribuidos otros modelos de computación distribuida pueden ser utilizados para segmentar sistemas sobre una red. El software intermedio orientado a mensajes(MOM de la lengua inglesa *Messaging-oriented middleware*), actúa similarmente a la invocación de métodos remotos tradicionales, pero con una variación. MOM, consiste en un conjunto de mensajes que son gestionados por una o más "oficinas postales", las cuales vuelven a enviar estos mensajes a los grupos de interés específicos. Este modelo tiene la ventaja de que puede soportar sistemas con clientes desconectados temporalmente tales como computadoras portátiles.

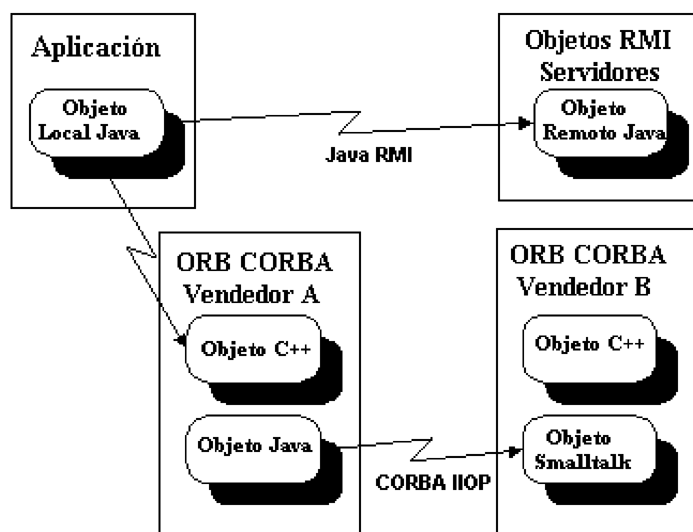


Figura 7.1.1 - Metodologías de Objetos Distribuidos

Los agentes móviles difieren de ambos de estos mecanismos de distribución. Con agentes móviles, el contenido ejecutable es trasladado a través de la red. Con relación a esto, un agente móvil puede ser considerado como un objeto distribuido que se mueve. La facilidad común de agentes móviles de OMG está en el proceso de determinar cómo los agentes pueden ser introducidos al mundo de CORBA. Se espera que el trabajo de OMG así como los otros esquemas estándar, terminen en un conjunto de conceptos comunes, mecanismos de comunicación y protocolos para aplicaciones basadas en agentes.

El decidir qué mecanismo de distribución utilizar depende de la aplicación que intentemos desarrollar. Cada una ofrece ventajas sobre las otras para un problema y dominio específicos. No obstante, pueden fusionarse más de una de las tecnologías para lograr resolver un problema de mayor envergadura.

7.1.2 Agentes móviles vs applets

Los agentes móviles heredan el modelo del código móvil entre redes hecho famoso por los applet de Java. Como una applet, los ficheros de clases para un agente pueden

migrar con él a través de una red. Pero a diferencia de los applets, un agente migra también con su estado. Un applet es un código que se puede mover a través de una red de un servidor a un cliente. Una agente es un programa que contiene código y estado y que puede moverse de un servidor a otro en una red. Adicionalmente, debido a que un agente lleva su estado a donde quiera que vaya, y puede viajar secuencialmente a muchos destinos en la red, incluyendo el regreso a su servidor original.

Un agente móvil es similar a un applet en que éste se ejecutara como un hilo de ejecución (thread) o múltiples hilos dentro del contexto de la aplicación del servidor. Para ejecutar un applet, un navegador del Web lanza una aplicación Java para recibir a cualquier applet que se encuentre, al moverse de una página a otra. Esta aplicación instala un gestor de seguridad para restringir las actividades de cualquier applet remoto. Para bajar los ficheros de clase de un applet, la aplicación crea cargadores de clases que saben cómo pedir los ficheros de clase que un servidor HTTP.

A diferencia, un agente requiere un servidor de aplicaciones, un servidor de agentes, que debe estar ejecutándose en un ordenador antes de que el agente pueda visitarlo. Cuando una agente viaja a través de la red, ellos migran de un servidor de agentes a otro. Cada servidor de agentes instala un gestor de seguridad para restringir las actividades de los agentes remotos. Los servidores reinician a los agentes a través de cargadores de clases que saben cómo recuperar los ficheros de clases y estado del agente provenientes de un servidor de agentes remoto.

7.2 VENTAJAS DE LOS AGENTES MÓVILES: SIETE BUENAS RAZONES PARA USARLOS

El interés en agentes móviles es motivado por los beneficios que proveen para la creación de sistemas distribuidos. Podemos considerar que los beneficios fundamentales que ofrece la tecnología de agentes móviles son: la reducción del tráfico en la red, la capacidad de desarrollar aplicaciones autónomas y la facilidad que ofrecen para crear aplicaciones que se ejecuten en paralelo y sean más rápidas.

Sin embargo, los agentes móviles no sólo pueden ofrecernos dichas ventajas, en ellos esconden más beneficios que se pueden descubrir al llevar a cabo implementaciones. Así que se pueden mencionar siete muy buenas razones para comenzar a usar esta tecnología [LANG98].

1. *Ellos reducen el tráfico de la red:* los sistemas distribuidos con frecuencia se basan en los protocolos de comunicación que involucran múltiples interacciones para lograr una tarea dada. Esto es especialmente verdad cuando las medidas de seguridad son habilitadas. El resultado es una gran cantidad de tráfico en la red. Los agentes móviles nos permiten empacar una conversación y enviarla a un servidor destino donde las interacciones pueden tomar lugar localmente. Los agentes móviles son también útiles cuando vienen a reducir el flujo de registros de datos en las redes. Cuando grandes cantidades de volúmenes de datos son almacenadas en servidores remotos, estos datos deberían de ser procesador en la localidad de los datos, en ves de transferirlos a través de la red. El fin es simple: mover la computación a los datos

en vez de los datos a la computación. Aunado a esto debemos mencionar que en los últimos años ha habido un crecimiento exponencial del número de usuarios en Internet (ver figura 7.2.1), los agentes móviles de igual forma contribuirán a reducir el número de interacciones dentro de la red mundial, lo que ayudará a que sea más rápida.

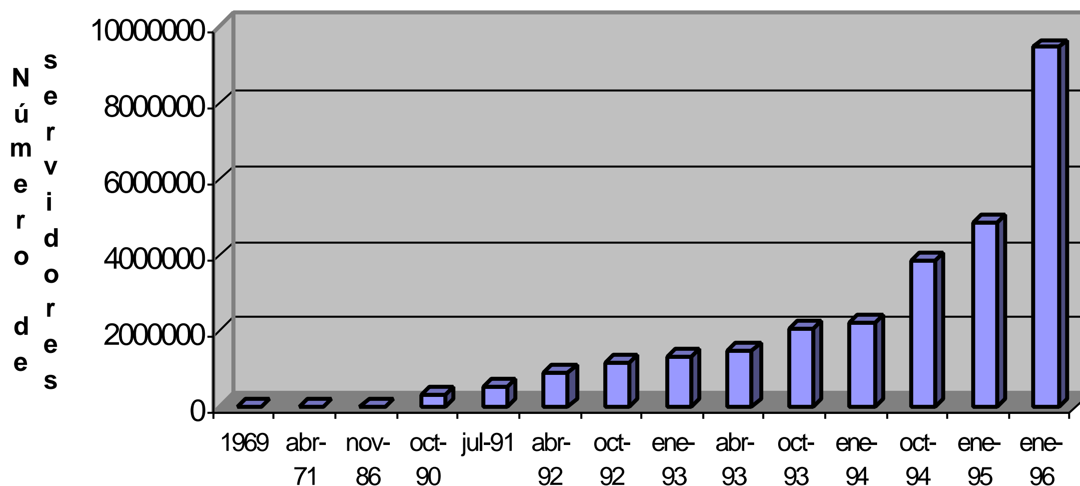


Figura 7.2.1 - Incremento del número de usuarios en Internet

2. *Ellos superan el estado latente de las redes:* los sistemas críticos de tiempo real tales como robots en procesos de manufactura necesitan responder a cambios en su entorno en tiempo real. Controlando tales sistemas a través de una fabrica en red de un tamaño sustancial implica estados latentes significantes. Para sistemas de tiempo real críticos, tales estados latentes no son aceptados. Loa agentes móviles ofrecen una solución, desde que ellos pueden ser despachados de un controlador central para actuar localmente y ejecutar directamente las direcciones del controlador.

3. *Ellos encapsulan protocolos:* cuando los datos son intercambiados en sistemas distribuidos, cada servidor posee el código que implementa los protocolos requeridos para codificar propiamente los datos de salida e interpretar los datos de entrada respectivamente. Sin embargo, como los protocolos evolucionan para acondicionar nuevas eficiencias o requerimientos de seguridad, es muy difícil sino es que imposible tarea la de actualizar el código del protocolo apropiadamente. El resultado es frecuentemente que los protocolos llegan a ser un problema de herencia. Loa agentes móviles, por otro lado, son capaces de moverse a servidores remotos con el fin de establecer canales basados en protocolos propietarios.

4. *Ellos se ejecutan de forma asíncrona y autónoma:* frecuentemente los dispositivos móviles tienen que depender de conexiones de red caras o frágiles. Esto es, tareas que requieren una conexión abierta continuamente entre un dispositivo móvil y una red no serán en la mayoría de los casos económica o técnicamente factible. Las tareas pueden ser incrustadas en agentes móviles, los cuales pueden ser despachados

hacia la red. Después de ser enviados los agentes móviles llegan a ser independientes de la creación de procesos y pueden operar de forma asíncrona y autónoma. El dispositivo móvil puede conectarse nuevamente más tarde para colectar al agente.

5. *Ellos se adaptan dinámicamente:* Los agentes móviles tienen la habilidad de percibir su entorno y reaccionar de forma autónoma a cambios. Un conjunto de agentes móviles posee la habilidad única de distribuirse a sí mismos entre los servidores en una red de tal forma que mantiene la configuración óptima para la solución de un problema particular.

6. *Ellos son heterogéneos por naturaleza:* la computación en redes es fundamentalmente heterogénea, con frecuencia desde las perspectivas de hardware y software. Como los agentes móviles son generalmente independientes del ordenador y de la capa de transporte, y dependientes sólo de su entorno de ejecución, ellos proporcionar condiciones óptimas para una integración del sistema transparente.

7. *Ellos son robustos y tolerables a fallas:* la habilidad de los agentes móviles para reaccionar dinámicamente a situaciones desfavorables y eventos hace más fácil el construir sistemas distribuidos robustos y tolerables a fallas. Si un servidor está siendo apagado, todos los agentes que se ejecutan en ese servidor serán avisados y se les dará tiempo suficiente para que se envíen a sí mismos y continúen sus operaciones en otro servidor de la red.

7.3 APLICACIONES DE LOS AGENTES MÓVILES

Los agentes móviles pueden ser utilizados para desarrollar una gran cantidad aplicaciones, estas aplicaciones pueden ser basadas en la tecnología de agentes, o bien pueden ser complementos de aplicaciones basadas en las tecnologías de orientación a objetos. Es decir, puede haber un sistema convencional que para realizar búsquedas remotas utilice la tecnología de agentes móviles.

De una u otra forma podemos encontrar que entre las aplicaciones más importantes que se pueden llevar a cabo con agentes móviles podemos citar[VENN97]:

- *Recolección de datos de distintos sitios:* una de las mayores diferencias entre el código móvil, tales como los applets, y los agentes móviles es el itinerario. Mientras que el código móvil usualmente viaja sólo de un punto a otro, los agentes móviles tienen un itinerario y pueden viajar secuencialmente a muchos servidores. De aquí que una aplicación natural de los agentes móviles, es la recolección de información a través de muchas computadoras enlazadas a una red. Un ejemplo de esta clase de aplicación es una herramienta de copias de seguridad que periódicamente debe supervisar cada disco instalado en cada computadora que se encuentra enlazada a la red. Aquí, un agente móvil podría navegar la red, recolectar la información acerca del estado de la copia de seguridad de cada disco y entonces regresar a su posición de origen y hacer un reporte.

- *Búsqueda y filtrado*: dado el constante incremento en la cantidad de información disponible en Internet y en otras redes, la actividad de recolectar información de una red implica la búsqueda entre grandes cantidades datos de unas cuantas piezas relevantes información. Eliminar la información irrelevante puede ser un proceso que consume mucho tiempo y quizá frustrante. En nombre de un usuario, un agente móvil puede visitar muchos servidores, buscar a través de la información disponible en cada servidor, y construir un índice de enlaces a las piezas de información que concuerdan con el criterio de búsqueda. El filtrado y la búsqueda muestran un atributo común a muchas aplicaciones potenciales de agentes móviles: el conocimiento de las preferencias del usuario.
  Aunque los agentes móviles no tienen que ser representativos o inteligentes, ellos normalmente lo son. De aquí, a un agente le es ha dado el conocimiento de las preferencias del usuario en términos de un criterio de búsqueda así como un itinerario, entonces es enviado a la red en nombre del usuario. Indaga a través de grandes cantidades datos por aquella información de particular interés para el usuario. En un momento, él regresa al usuario en reporte de sus hallazgos.

- *Monitorización*: en algunas ocasiones la información no está distribuida a través de un espacio como puede ser un conjunto de ordenadores, sino a través del tiempo. Nueva información constantemente está siendo generada y publicada en la red. Los agentes pueden ser enviados para esperar por ciertas clases información hasta que ésta sea generada o se encuentre disponible.
  Por ejemplo un agente personalizado para la reunión de noticias. Una gente podría monitorizar varias fuentes de noticias para determinado tipo de información de interés para su usuario, entonces informarle cuando información relevante esté disponible. Otro ejemplo puede ser las aplicaciones de monitorización de servidores de red en donde mediante agentes móviles se monitorea el consumo de recursos y el estado general del sistema, para profundizar en estas aplicaciones refiérase a [PEREb98a] y [PEREb98b].
  Esta aplicación resalta la naturaleza asíncrona de los agentes móviles. Si se envía un agente, no es necesario sentarse y esperar por resultados de la información recopilada. Se puede programar un agente para que espere todo lo que sea necesario hasta que cierta información se encuentre disponible. También no se requiere permanecer conectado a la red hasta que el agente regrese. Un agente puede esperar hasta que su usuario se reconecte a la red antes de realizar un informe para él.

- *Comercio electrónico:* El comercio electrónico es otra aplicación apropiada para el uso de la tecnología de agentes móviles. Un agente móvil podría realizar compras, incluyendo la realización de ordenes de compra y potencialmente pagar. Por ejemplo, si se quiere volar de un sitio a otro, un agente podría visitar las bases datos de los horarios de vuelo y los precios de varias líneas aéreas, encontrar el mejor precio y horario de salida, hacer la reserva acción e inclusive pagar con un número de tarjeta de crédito.
  El comercio electrónico también puede llevarse a cabo entre agentes. Por ejemplo podría haber un servidor de agentes dedicado a la compra y venta de automóviles. Si se desea comprar un coche, se le podría dar a un agente un conocimiento que especifique las preferencias del usuario, incluyendo el rango de precios y una potencial estrategia de negociación. Entonces se enviaría al agente al servidor

dedicado, donde el se reuniría con otros agentes que están buscando vender un coche.
Si una oportunidad potencial fuera encontrada, el agente podría informar para que se contactara a la persona indicada y realizar los arreglos finales. Alternativamente, el agente podría consumar el trato en su nombre, sobre todo en el caso de que la oportunidad sea única y en la cual sólo tenga unos segundos para tomarla decisión sin que otro agente compre el coche primero.

- *Procesamiento paralelo*: dado que los agentes móviles pueden moverse de un modo otro y pueden reproducirse, un uso potencial de la tecnología de agentes móviles es una forma de administrar aplicaciones que requieran procesos en paralelo. Ciertamente prácticamente todas las aplicaciones descritas en estas líneas pueden hacer uso de la concurrencia para mejorar enormemente su eficiencia y lograr de esta manera aplicaciones más rápidas.

- *Distribución masiva información*: otro uso potencial de los agentes móviles es la distribución interactiva de noticias o publicidad a grupos de interés. Los agentes móviles pueden usarse en este caso al igual que el correo electrónico para distribuir indiscriminadamente información si se trata de publicidad, o bien especificarle al agente el tipo de servidor en donde habrá de distribuir dicha información en el caso de ser información mas privada.

- *Interacción y negociación:* además de la búsqueda en bases datos y ficheros, los agentes pueden obtener información mediante la interacción con otros agentes. Si por ejemplo, se desea realizar una agenda de reuniones con otras personas, se puede enviar a un agente móvil para que interaccione con agentes representantes de cada una de las personas que se desea en invitar a la reunión. Los agentes pueden negociar para establecer el horario de la reunión. En este caso, cada agente contiene información acerca de la agenda de su usuario. Para lograr el acuerdo del tiempo establecido para la reunión, los agentes necesariamente necesitan intercambiar información.

- *Supercomputadora virtual (cálculos en multiprocesos):* los cálculos complejos con frecuencia pueden ser descompuestos en unidades discretas para la distribución en una pila de servidores o procesos. Cada una de estas unidades discretas puede ser asignada a un agente, el cual es entonces enviado a un sitio remoto en donde el trabajo es realizado. Una vez terminado, cada agente puede regresar a casa con los resultados los cuales pueden ser agregados y sumados.

- *Redes parcialmente desconectadas*: en sistemas con redes frecuentemente desconectadas, los agentes pueden salvar los procesos de la red al mover el contenido ejecutable hacia los datos origen en vez de intentar constantemente conexiones de red a los datos origen.

- *Entretenimiento*: un último ejemplo de las aplicaciones potenciales para los agentes móviles es el entretenimiento. En este escenario, los agentes representan a los jugadores del juego. Los agentes compiten con otros en representación de los jugadores. Cada jugador programada un agente con una estrategia, entonces enviará al agente al servidor del juego. Esto facilita que los jugadores sean remotos, y si el

servidor del juego se ejecutara en un ordenador en Las Vegas, entonces quizá los agentes podrían jugar con dinero real.

- *Entrega de correo inteligentemente:* los agentes podrían entregar un correo electrónico basándose en el contenido del mensaje. Para lograrlo el agente tendría que llevar en su código las reglas de decisión.

Para profundizar mas en las ventajas que ofrece la tecnología de agentes móviles así como en las aplicaciones que se pueden implementar con ellos refiérase a [CHES95].

7.3.1 Un ejemplo de consultas a Sistemas de Información Geográfica (SIG) mediante agentes móviles

Supongamos que existe una autopista entre las comunidades de Asturias y Madrid la cual atraviesa la comunidad de Castilla y León, dicha autopista pasa por las ciudades más importantes de las comunidades, sin embargo las poblaciones medianas y muy pequeñas no cuentan con tramos de autopista que le permitan enlazar directamente con la autopista Asturias y Madrid.

Nuestro problema consiste en detectar todas aquellas poblaciones que sean mayores a cuarenta mil habitantes y cuya distancia de la autopista no exceda de treinta kilómetros [CUEV98b]. El plan consiste, en que una vez detectadas dichas ciudades, se construya un tramo de autopista que enlace con la autopista principal Asturias-Madrid a dichas ciudades. Obsérvese la siguiente figura.

Figura 7.3.1 - Autopista Asturias - Madrid

Para implementar dicha tarea habrán servidores de SIG y de agentes móviles en cada una de las comunidades en donde se hará una recopilación parcial de información (tanto

alfanumérica: distancias de los núcleos de población, como gráfica: perfil de cada tramo a realizar), a su vez el agente que llegue a cada comunidad podrá crear agentes hijos que enviará a otras ciudades pequeñas a que busquen información de las poblaciones más cercanas a ellas (Vea figura 7.3.2).

A su vez el agente, llevará consigo la información del trazado de la autopista que se desea realizar así como de las restricciones de las poblaciones que se desean unir con la autopista, además de la lógica necesaria para poder procesar la información y regresar sólo con la información necesaria y ya filtrada.

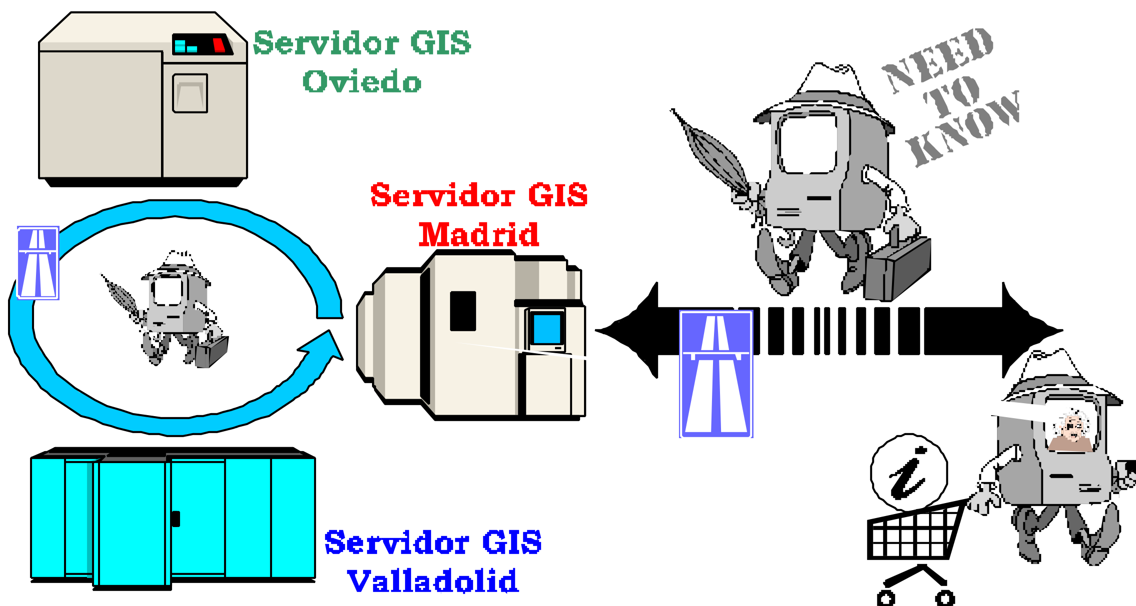


Figura 7.3.2 - Servidores de recopilación parcial de información

La consulta podría realizarse desde fuera del país. Supongamos que se envía un agente desde exterior a España a hacer dicha tarea, dicho gente será recibido en Madrid y creará 3 agentes hijos que se encarguen de recopilar la información en cada una de las comunidades de Asturias, Castilla y León y Madrid. Una vez creados éstos agentes serán enviados a cada una de las comunidades en donde pasarán por un proceso de verificación de seguridad.

Ahora profundicemos en detalle en una parte de nuestro modelo. El agente que llega a la comunidad de Castilla y León será recibido en Valladolid, creará a su vez otros tres agentes que se encargarán de recopilar información parcial de las ciudades por donde pasa dicha autopista, enviando nuevos agentes a las ciudades de León, Valladolid y Segovia.

Estos agentes serán los que en verdad realizaran las tareas de recopilación, una vez que lleguen a sus destinos y sean verificados por el gestor de seguridad, establecerán el protocolo de intercambió de información que usarán con el sistema información geográfica. Dicho protocolo podrá ser a través del JDBC o bien considerando que el SIG sea visto como un control ActiveX. Los agentes llevarán consigo la información que indica la búsqueda de poblaciones mayores a cuarenta mil habitantes y que se encuentren a no más de treinta kilómetros de la autopista.

De igual forma este proceso se realizará concurrentemente en las otras provincias con el fin de realizar la búsqueda de una forma más rápida, de tal manera que en cada provincia el agente padre creará agentes hijos que se distribuyan en otras ciudades.

Una vez enviada la información, el SIG procesará internamente la consulta y devolverá los resultados. Entonces los agentes regresaran al sitio de donde fueron enviados y entregarán los resultados al padre que será el encargado de recopilarlos y formar un resultado parcial de cada entidad (observe figura 7.3.3).

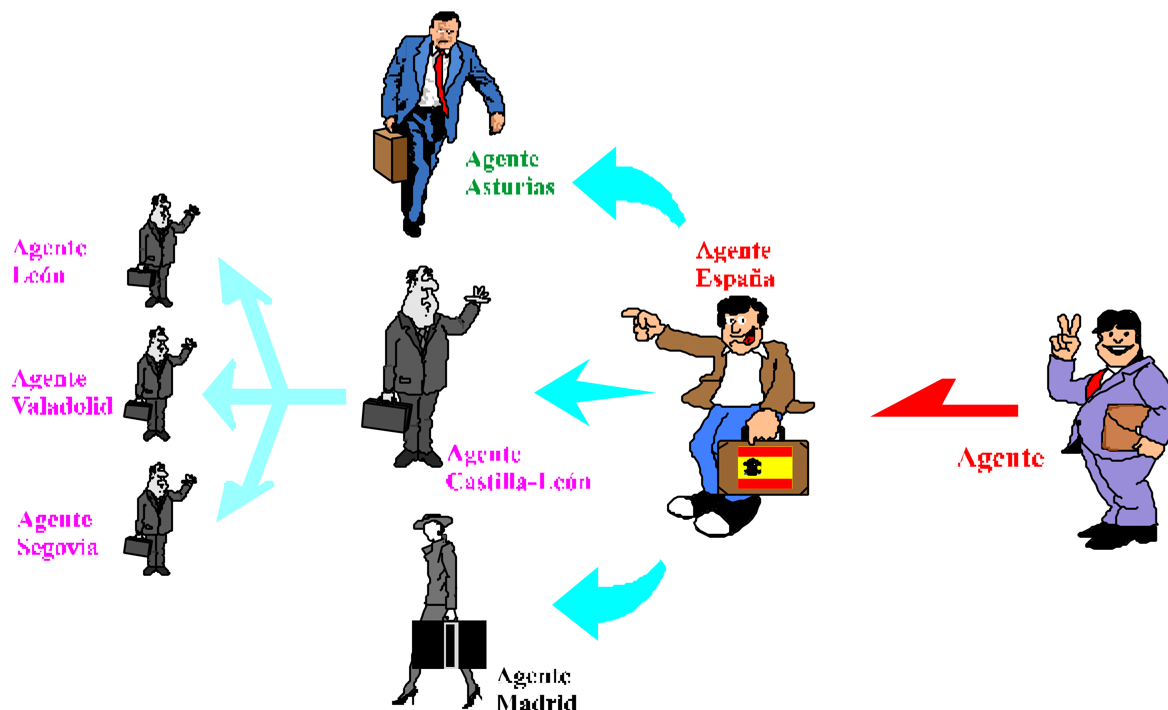


Figura 7.3.3 - Delegación de tareas entre agentes

Finalmente, los agentes padres de cada provincia, regresaran al agente principal que inicio la consulta, el cual se encuentra en Madrid. Así pues, todos los agentes con resultados parciales de cada autonomía regresarán a Madrid para que el agente principal pueda recopilar toda la información y finalmente regresar con todos los datos hacia el sitio en donde se haya iniciado la consulta.

Como resultado del proceso, en el origen el usuario podrá obtener una representación de los tramos de autopista a realizar e información precisa sobre las distancias de los mismos así como otros aspectos técnicos para la construcción de la autopista, como tipo y morfología de la geografía afectada por la obra.

Capítulo VIII

EL MUNDO DE LOS AGENTES MÓVILES

La aparición de las tecnologías de agentes móviles se ha convertido en uno de los acontecimientos más importantes que han tenido lugar en el mundo de la informática y de las redes de ordenadores en los últimos años. Esto se debe a que jugarán un papel primordial en el desarrollo de sistemas distribuidos, así como en la búsqueda y recopilación de información a través de redes locales e Internet. Por tanto darán lugar a una nueva generación de aplicaciones en los años venideros. Los agentes terminarán convirtiéndose en asistentes personales que podrán realizar tareas de forma autónoma en Internet en nombre de sus usuarios.

Actualmente una gran cantidad de empresas y universidades han iniciado proyectos de investigación para crear nuevas plataformas para desarrollar sistemas agentes de móviles. Hay quienes han apostado que el futuro de Internet está en esta tecnología y han desarrollado nuevos prototipos y esquemas de seguridad que satisfagan apropiadamente las necesidades que un sistema agentes móviles requiere, superando así su principal inconveniente.

A la vez se ha iniciado aplicaciones específicas utilizando las plataformas existentes sobre todo enfocadas a comercio electrónico y a la búsqueda y recopilación de datos dentro de una red. No obstante para que dichas aplicaciones tengan éxito y puedan llegar a convertirse en *'The killer application'* es necesario que las plataformas maduren y ofrezcan el soporte indispensable.

La llegada de Java al mercado causó una gran euforia dentro de la comunidad de agentes móviles. Las características que poseía ofrecían una gran esperanza de solucionar muchos de los problemas con los que otras plataformas se estaban enfrentando. Debido a esto se han desarrollado una gran cantidad de sistemas de agentes móviles basados en Java, que ya superan el número de sistemas para el desarrollo agentes móviles que existía antes de su llegada.

Aunque se han logrado avances considerables, la investigación en el campo de agentes móviles puede considerarse aún en sus inicios. Primero se tendrán que superar los dos grandes retos: **portabilidad y seguridad**, para posteriormente pensar en que las

plataformas ofrezcan entornos visuales para el desarrollo de agentes, directorios de páginas amarillas, interfaces gráficas integradas (para que los agentes interactúen con los usuarios) así como sistemas de depuración que permitan crear un entorno real para el desarrollo de aplicaciones con esta tecnología.

8.1 PORTABILIDAD Y SEGURIDAD

La portabilidad y la seguridad serán los principios fundamentales para los sistemas de agentes móviles. La portabilidad es una necesidad para que los agentes móviles sean capaces de moverse en redes heterogéneas (entre máquinas con diferentes sistemas operativos y arquitecturas de hardware) y puedan ser realmente útiles.

La seguridad es importante porque el servidor debe mantener efectivamente el control de un programa externo de efectos desconocidos. Existe también el problema inverso en la seguridad del agente, es decir contra acciones indeseadas por parte del servidor, como por ejemplo, espiar el contenido del agente. Sin embargo, no hay una solución general para este problema excepto algunos aspectos individuales.

La mayoría de los sistemas agentes móviles existentes, mientras difieren considerablemente en la práctica, usan la misma solución básica para la portabilidad y la seguridad. Ellos no ejecutan a los agentes en una máquina real, sino en una virtual, generalmente un intérprete y un sistema de *runtime*, los cuales esconden los detalles de la arquitectura del servidor agentes y encierran las acciones de los agentes a ese entorno restrictivo.

8.1.1 La necesidad de una máquina virtual

La característica fundamental de las máquinas virtuales es que su principal destino es la implementación software[ALVA98]. Aunque no se excluye que se puedan implementar mediante hardware, la certeza de que existirán mediante software elimina los inconvenientes ligados a la naturaleza hardware de las máquinas reales.

No es necesario sustituir el hardware existente con anterioridad para utilizarlas. Basta desarrollar el simulador software necesario para cada plataforma. De esta manera se pueden utilizar en entornos heterogéneos sin dificultad, y el código binario de la máquina virtual puede funcionar en cualquier plataforma (en la que se haya implementado el simulador). Además se tiene la flexibilidad del software que permite modificar fácilmente decisiones de implementación, añadir o eliminar características, etc.

Es evidente que los sistemas de agentes móviles deben ser ejecutados sobre una máquina virtual que asegure la portabilidad a diversas plataformas y permita que los agentes sean ubicuos, de igual forma, mediante un interprete será más fácil transportar su estado de ejecución así como realizar las verificaciones de seguridad necesarias que garanticen la integridad del sistema. Presentaré a continuación algunas de las principales plataformas para el desarrollo de agentes.

8.2 PLATAFORMAS PARA EL DESARROLLO DE AGENTES MÓVILES

Desde los principios de esta tecnología se han creado diversas plataformas para el desarrollo de agentes móviles. Las primeras quizá carecieron de muchos requerimientos que exigen las aplicaciones reales, tal vez fue la razón por la que no tuvieron éxito, sin embargo sentaron las bases y los conceptos de esta tecnología que cada día ha ido mejorando.

Existe una gran cantidad de plataformas para el desarrollo agentes móviles, todas, si bien siguen la misma filosofía, difieren en la creación de agentes así como en los protocolos de transmisión. Por razones de espacio haré el análisis de las plataformas de agentes móviles más importantes que existieron o bien se están desarrollando en nuestros días.

Las nuevas plataformas copian las características que ofrecían las anteriores y añaden nuevos conceptos que facilitan el desarrollo de las aplicaciones, la aparición de Java originó un desarrollo más rápido de la tecnología de agentes incorporando al mercado una gran cantidad de sistemas de agentes móviles basados en Java de los que presentaré los más importantes.

8.2.1 Telescript

La primera implementación comercial del concepto de agentes móviles, fue la tecnología de Telescript de General Magic[15], esta tecnología intentó permitir acceso automático e interactivo a las redes de ordenadores usando agentes móviles. El enfoque comercial de la tecnología de General Magic fue el comercio electrónico, que requería que una red permitiera a los proveedores y consumidores de servicios encontrarse uno al otro y hacer negocios electrónicamente.

Los creadores de Telescript apreciaron al comercio electrónico como una pequeña pieza del mundo de agentes móviles que existirá en los próximos años. Telescript implementa una arquitectura que abarca diversos módulos y conceptos[16] que le permiten crear sistemas asociados con programación remota.

La arquitectura incluye una división de módulos que favorecen su objetivo principal, el comercio electrónico, así pues se incluyen lugares, puntos de reunión, autoridades, agentes, conexiones y permisos que son necesarios para que un sistema agentes móviles pueda resolver dicha tarea.

Para facilitar el desarrollo de aplicaciones de comunicación, los agentes y los lugares son escritos en el lenguaje de Telescrip, el cual posee las siguientes características:

- Es un lenguaje completo.

[15] Compañía que más tarde abandonaría Telescript por iniciar un nuevo sistema de agentes móviles basado en Java llamado Odissey.
[16] Los que fueron vistos en la sección 6.5.

- Orientado a objetos.
- Dinámico.
- Persistente
- Portable y seguro.

Además del lenguaje de programación, Telescript provee un servidor (llamado máquina en el lenguaje de Telescript) que es un programa de software que implementa el lenguaje para mantener y ejecutar los lugares dentro de su contexto, al igual que a los agentes que ocupan esos lugares.

El lenguaje ofrecía la mayor parte de los requerimientos que requería un agente en el comercio electrónico: seguridad, movilidad y transacciones. Telescript y sus tecnologías asociadas son ampliamente reconocidas como los fundadores de las bases, filosófica y técnicamente, para la mayoría de los otros sistemas de agentes móviles que se usan hoy en día, y en realidad de al menos todos los que analizaremos en este capítulo.

La plataforma de Telescript fue precursora de los sistemas de agentes móviles, quizá su único problema por el que no tuvo éxito fue su incompatibilidad con algún lenguaje de programación existente, lo que implicaba que los usuarios tuvieran que aprender un nuevo lenguaje de programación para desarrollar sistemas de agentes. Aunado a esto debemos añadir una mala campaña de mercadotecnia. Como Telescript se considera muerto[17] actualmente, no profundizaré en él en lo que resta del capítulo.

8.2.2 D'Agents

En términos generales D'Agents posee cuatro características principales. Les facilita a los agentes el navegar al reducir la migración a una simple instrucción *agent_jump*, la cual puede ser usadas en cualquier punto de su operación. Provee una forma de comunicación flexible y de bajo nivel. Proporciona un lenguaje de codificación de alto nivel como lenguaje principal. Y finalmente proveen un modelo de seguridad efectivo, el cual es necesario para su aceptación en el mundo de los agentes.

Adicionalmente a su arquitectura bien desarrollada la cual sienta las bases para sistema de D'Agents, otra ventaja del sistema viene del uso de su lenguaje de codificación *Tcl*, que permite incorporar extensiones como *Tk*, que le permitirán al agente trabajar con interfaces gráficas. Esta plataforma es ampliamente analizada en la sección 8.3.

8.2.3 Ara: Agentes para acceso remoto

El principio fundamental de esta arquitectura consiste en separar el núcleo que implementa todas las operaciones para llevar a cabo la gestión de agentes de los interpretes del lenguaje. La arquitectura de Ara permite la incorporación de cualquier lenguaje al sistema, basta con crear un interprete para ese lenguaje, de esta forma

[17] Ya que general Magic ha dejado de darle soporte.

permite que los agentes sean programados en más de un lenguaje de programación, lo que representa la filosofía principal de esta plataforma.

Ara pretende crear una plataforma para el desarrollo de agentes móviles que sea independiente de la máquina, sistema operativo y lenguaje que lo soporta. Adicionalmente, el sistema proporciona facilidades para los requerimientos específicos de los agentes móviles en aplicaciones reales, en donde la seguridad viene a ser el más prominente. Un análisis amplio de la plataforma se encuentra en la sección 8.4.

8.2.4 JAVA

Java fue concebido en *Sun Microsystems Inc.* 1991. El desarrollo de la primera versión duró 18 meses. El lenguaje se llamó inicialmente *Oak*, pero se le puso el nombre de Java en 1995.

La llave que permite a Java resolver los problemas de seguridad y portabilidad descritos anteriormente es que la salida de un compilador de Java no es ejecutable, sino que es un código binario [NAUG97]. Este código binario es un conjunto de instrucciones altamente optimizando diseñado para ser ejecutado por una máquina virtual que emula el intérprete de Java. Es decir, el intérprete de Java es un intérprete de código binario. Java fue diseñado para ser un lenguaje interpretado, ya que es mucho más fácil de ejecutar en una variedad de entornos distintos.

8.2.4.1 La máquina virtual y portabilidad

El éxito de la máquina virtual de Java, se debe principalmente a que es capaz de crear una plataforma homogénea para el desarrollo de aplicaciones que pueda instalarse sobre diversas plataformas heterogéneas. La emulación de la máquina virtual está disponible en prácticamente todas plataformas del mercado y permite así la ejecución de las clases Java bajo cualquier plataforma. De igual forma favorece a que muchas aplicaciones de movilidad y distribución puedan ser implementadas sobre ella.

El gran éxito de Java no es tanto debido al lenguaje sino al conjunto de tecnologías que le rodean. El elemento fundamental de este éxito es la asociación existente entre el lenguaje y una máquina virtual JVM (*Java Virtual Machine*) [ALVA98] creada especialmente para darle soporte. La portabilidad que proporciona la máquina virtual, junto con la coincidencia en el tiempo del lanzamiento de Java y la explosión de la Internet son las razones fundamentales de su éxito. Internet es una red heterogénea, y el uso de una máquina virtual es ideal en esas circunstancias: el código para la máquina virtual puede moverse por la red, independientemente de cuál sea la plataforma destino.

La máquina virtual de Java está totalmente adaptada para soportar el lenguaje Java: lenguaje Orientado Objetos con clases y herencia simple, herencia múltiple de interfaces, enlace dinámico, concurrencia basada en monitores, excepciones, etc.

8.2.4.2 Seguridad

Dado que uno de los objetivos es la utilización de código que pueda ser descargado de Internet, existen una serie de restricciones de seguridad que intentan evitar daños malintencionados. Por una parte, el componente que se encarga de cargar dinámicamente las nuevas clases en el área de métodos tiene un verificador de clases que comprueba que el código de la clase cumple una serie de condiciones para que sea seguro (por ejemplo que no intenta acceder fuera de su ámbito).

Por otro lado se puede asociar un gestor de seguridad a una determinada aplicación limitándole el acceso a un conjunto de recursos que no pueden causar daños al sistema. Por ejemplo se impide el acceso al sistema de ficheros general del sistema. Esta política de seguridad, denominada Caja de Arena (*Sandbox*), aunque efectiva, es demasiado restrictiva.

La nueva versión del JDK, permite las firmas digitales lo que permitirá crear un nuevo esquema de seguridad para los sistemas de agentes móviles.

8.2.4.3 Serialización de objetos, RMI y componentes

Usando la serialización de objetos las clases se pueden enviar por la red [CUEV98b], de esta forma es posible enviar tanto los datos como la computación. De igual forma la invocación de métodos remotos (RMI), permite acceder a métodos de objetos remotos con lo que se logra la interacción entre objetos que en un momento dado podrían solicitar la transferencia de otros objetos.

Otra característica adicional es que las clases se pueden agrupar en componentes (JavaBeans). Los JavaBeans han ganado rápidamente la atención del mercado, son independientes de la plataforma pero dependientes del lenguaje. Las capacidades de los componentes JavaBeans son implementadas como un conjunto de extensiones del lenguaje para la librería de clases de Java estándar. Así, los JavaBeans son un conjunto de interfaces del lenguaje de programación de Java. Éstos logran la portabilidad de la plataforma a través de la máquina virtual de Java.

8.2.4.4 Multihilo y Multitarea

También debemos tener en cuenta que Java es un lenguaje ubicuo y que es multihilo lo que permite crear un proceso independiente y autónomo para cada agente, lo que lo hace ideal para programar agentes móviles.

De esta forma es posible eliminar un agente al eliminar su hilo correspondiente sin afectar el resto del sistema, de igual forma permite que un hilo pueda crear mas hilos lo que facilita que los agentes puedan crear hijos. La creación de diversos hilos a la vez permite la ejecución de varias tareas en forma concurrente.

HERE -HERE

8.2.4.5 Todo listo para los sistemas de Agentes Móviles

El lenguaje Java se ha introducido muy rápidamente en la tecnología de agentes móviles. Con la infraestructura Java ya se dispone de casi todo lo necesario para crear agentes que puedan navegar por Internet.

Java ofrece todo lo que una plataforma de agentes móviles requiere: portabilidad, seguridad, serialización de objetos, RMI, ejecución de múltiples hilos concurrentemente y ubicuidad.

La existencia en Java de todas estas características es la razón principal por el que muchos de los proyectos que ofrecen una infraestructura para agentes móviles se estén implementados en Java, entre los más importantes podríamos mencionar:

- IBM Aglets.
- MOLE.
- Concordia.
- Voyager.

Por razones de espacio en este capítulo se realizará el análisis de los más importantes.

8.3 AGENT TCL O D'AGENTS

Agent TCL es un poderoso sistemas agentes de Internet que corre en estaciones de trabajo UNIX que permite el desarrollo rápido de agentes complejos [COCK98]. *Agent TCL* es una plataforma efectiva para la experimentación de agentes en Internet y para el desarrollo de aplicaciones de pequeño y mediano alcance. Los agentes de *Agent TCL* que están inscritos en una versión extendida del lenguaje de comandos de herramientas TCL (de la lengua inglesa *Tool Command Lenguage*) desarrollado por *Sun Microsystems.* Que es un lenguaje de codificación de alto nivel y que es a la vez poderoso y fácil de aprender.

Los agentes de *Agent TCL* pueden usar todos los comandos estándar de TCL así como un conjunto de comandos especiales que son proveídos cómo una extensión a TCL. Estos comandos especiales permiten a un agente migrar de una máquina a otra, crear agentes hijos, comunicarse con otros agentes, así como obtener información acerca de la localización actual en la red de los agentes. Adicionalmente, *Agent TCL* puede ser extendido con un conjunto de comandos definidos por el usuario para crear un sistema de agentes más poderoso.

El nombre de *Agent TCL* fue modificado [DARTa] a *D'Agents* en la última versión del producto, al considerar la modificación de su núcleo para que el producto ofreciera la capacidad de soportar diversos lenguajes para el desarrollo agentes (Tcl, Java y Python). De aquí que sus autores consideraran que ya no debería de llamarse *Agent TCL*, y hayan decidido un nombre más genérico es decir: *D'Agents.*

En lo que resta de este capítulo haré referencia a *Agent TCL* como *D'Agents.*

8.3.1 Generalidades de D'Agents

Antes de profundizar en cuestiones de arquitectura y seguridad, hablaré de las características fundamentales que un sistema de agentes debe tener como lo son la movilidad y la comunicación, así como otras características que ofrece este sistema de agentes.

8.3.1.1 Movilidad de agentes

En *D'Agents*, la migración es lograda con el comando *agent_jump*, el cual puede aparecer en cualquier lugar dentro del agente. Éste captura el estado actual del agente y transfiere una imagen de ese estado a un servidor en la máquina destino. En otras palabras, permiten que el agente suspenda su ejecución en un punto arbitrario, se transporte a otra máquina, y continúe su ejecución en la nueva máquina en el punto exacto en el cual la dejo. Una vez que ha emigrado puede acceder a los recursos del sistema así como comunicarse con otros agentes.

8.3.1.2 Comunicación entre agentes

Existen dos formas de comunicación entre agentes. La primera es el paso de mensajes, la cual usa los conceptos tradicionales de enviar y recibir. La segunda forma de comunicación es una conexión directa, la cual es esencialmente un flujo de mensajes nombrados. Un agente establece una conexión directa con otro agente usando el comando *agent_meet*. Los dos agentes entonces intercambian mensajes a través de la conexión. Un mensaje en *D'Agents* es una cadena de caracteres sin una sintaxis o semántica bien definida, de tal forma que los agentes deben acordar el significado de los mensajes que ellos transfieren.

8.3.1.3 Otras características de D'Agents

Otros comandos permiten a un agente crear agentes hijos y obtener información acerca del estado actual de la máquina, tales como la identidad de otros agentes. Adicionalmente, los agentes pueden usar la librería de herramientas *Tk* (del inglés *Tool Kit)* para interactuar con el usuario de la máquina actual. *Tk* es una herramienta para crear interfaces gráficas de usuario para programas en TCL. Debido a que permite a una interfaz gráfica de usuario sea descrita completamente en TCL, interfaces con calidad profesional pueden ser creadas con una relativa pequeña cantidad de tiempo y código. Un agente que quiere interactuar con un usuario durante el curso de su viaje podría llevar el código necesario *Tk* con él.

8.3.2 Objetivos de la arquitectura de D'Agents

D'Agents tiene cuatro objetivos fundamentales:

- Reducir la migración a una simple instrucción, y permitir que esta instrucción pueda ocurrir en cualquier punto arbitrario del código del agente. Con lo que se logra una transmisión real del estado de ejecución.
- Proveer mecanismos de comunicación que sean flexibles y de bajo nivel, pero que escondan los detalles de la transmisión.
- Proveer un lenguaje de codificación que sea de alto nivel, pero que soporte múltiples lenguajes y mecanismos de transporte, permitiendo la adición directa de un nuevo lenguaje o mecanismo de transporte. La facilidad de lenguajes múltiples es particularmente importante para aquellos agentes que requieran grandes cantidades de código o que ejecuten tareas críticas en velocidad en donde TCL no sea apropiado .
- Proveer una seguridad efectiva en el incierto mundo de Internet.

8.3.3 La arquitectura de D'Agents

Esta arquitectura está construida sobre el modelo servidor de Telescript, los lenguajes múltiples de ARA, y el mecanismo de transporte de dos sistemas predecesores en Dartmouth. La arquitectura consta de cuatro niveles, obsérvese la figura 8.3.3.1. El nivel más bajo es una interfaz de programación de aplicaciones(API) para cada mecanismo de transporte disponible [GRAY95b].

El segundo nivel es un servidor que se ejecutará en cada servidor de la red a donde los agentes quieran ser enviados. El servidor realiza las siguientes tareas:

- *Estatus*: el servidor mantiene un registro de los agentes que están ejecutándose en esa máquina y contesta preguntas acerca de su estado.
- *Migración*: el servidor acepta a cada mensaje que llega, autentifica la identidad del propietario, y pasa el agente autentificado al intérprete apropiado para su ejecución.
- *Comunicación:* el servidor provee un espacio de nombres jerárquico para los agentes y para permitir que los agentes se envíen mensajes unos a otros dentro de ese espacio de nombres. La división más alta del espacio de nombres es el nombre simbólico de localización de la red del agente.
- *Almacenamiento no volátil:* el servidor proporciona acceso a un almacenamiento no volátil para que los agentes puedan realizar copias de seguridad de su estado interno cuando lo deseen.

El resto de los servicios son proporcionados por los agentes. Tales servicios incluyen descubrimiento de recursos, comunicación de grupos, tolerancia a fallos, control de acceso, y comunicación independiente de la localización. Sin embargo los agentes de servicios más importantes en este prototipo son los agentes de reenvío y los agentes de gestión de recursos. Los agentes de reenvío soportan operaciones de desconexión, si un agente no es capaz de migrar a la localización deseada por que la máquina o red fallan, el agente es añadido a una cola o lugar de reenvío dentro de la red. El agente de envío dirige al agente a la localización deseada hasta que le sea posible alcanzarla. Los agentes gestores de recursos, en combinación con el sistema de encriptación de muy buena seguridad PGP (del inglés *pretty good privacity)* y con los módulos de seguridad de lenguaje específico como *Safe-Tcl*, guardan el acceso a recursos críticos del sistema

tales como el disco. Hablaré más en detalle sobre estos aspectos en la parte de seguridad.

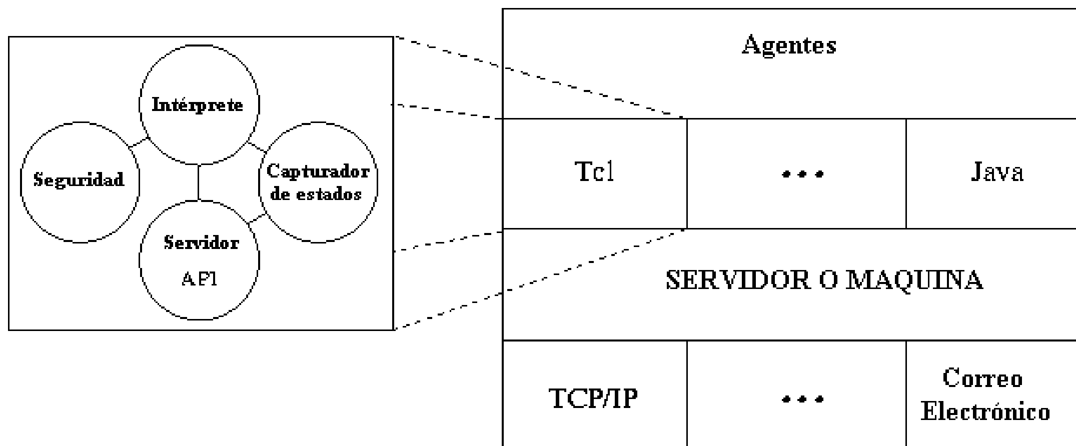


Figura 8.3.3.1 - La arquitectura de D'Agents

La tercera capa de la arquitectura consiste en un intérprete para cada lenguaje disponible. Cada intérprete tiene cuatro componentes: El intérprete mismo, un módulo de seguridad que previene al agente de realizar acciones maliciosas, un modelo de estado que captura y restablece el estado interno de un agente en ejecución, y una API que interacciona con el servidor para gestionar la migración, comunicación y la posición del puntero de ejecución. Al añadir un nuevo lenguaje es necesario crear todos los componentes arriba mencionados para ese lenguaje, excepto el de seguridad, ya que este modulo solo asegura que un agente no pasa la gestión de recursos o viola las restricciones de seguridad impuestas por el gestor.

El nivel más alto de la arquitectura de *D'Agents* consiste en los mismos agentes.

8.3.4 El lenguaje de desarrollo de D'Agents

D'Agents se basa en el lenguaje de codificación TCL para el desarrollo de agentes. TCL es un lenguaje de codificación de alto nivel que fue desarrollado en 1987 por *Sun Microsystems.*

Un agente en D'Agents es un código que se ejecutan sobre un intérprete modificado y una extensión de Tcl. La extensión provee un conjunto de comandos que el código usa para migrar, comunicar y crear agentes hijos [GRAY95a]. Estos comandos aunque son proveídos como una extensión de Tcl, pueden ser tratados como una parte nativa del lenguaje al escribir un agente, estos comandos pueden ser agrupados en tres categorías que presentaremos a continuación.

8.3.4.1 Comandos para la creación de agentes

- *Agent_begin:* este comando registra al agente con un servidor de agentes en la máquina especificada. Regresa un nuevo identificador del agente dentro del espacio de nombres del agente.
- Los paréntesis redondos vacíos indican que el agente no tiene un nombre simbólico inicialmente, el nombre simbólico puede ser escogido más tarde mediante el comando *agent_name.*
- *agent_name:* el comando *agent_name* selecciona un nombre simbólico para el agente .
- *agent_end:* un agente llama a este comando cuando éste finaliza con sus tareas y no requiere ningún servicio más.

8.3.4.2 Comandos para la migración de agentes

- *agent_jump:* este comando es utilizado cuando un agente quiere migrar a una nueva máquina. El comando captura el estado interno del agente y envía el estado a un servidor de agentes en una máquina especifica. El servidor restablece el estado de ejecución del agente inmediatamente después de *agent_jump*.
- *agent_fork:* crear una copia del agente en una máquina especifica. Tanto el agente original como la copia continúan su ejecución en el lugar en donde se ejecutó el comando.
- *Agent_submit:* Este comando crea una compañía diferente de un nuevo agente. Los parámetros que se le transmiten son la máquina, una lista de variables, una lista de procedimientos, y un programa de inicio de ejecución.

8.3.4.3 Comandos para la comunicación de agentes

- *agent_send:* este el comando envía un mensaje a otro agente. El mensaje consiste de un código entero y un string arbitrario. El receptor recibe el mensaje mediante el comando *agent_receive*, o bien se está usando Tk, estableciendo un gestor de eventos para los mensajes que llegan de otros sitios.
- *agent_event*: es una variante del comando anterior y la diferencia es que envía una etiqueta y una cadena en vez de un código entero y una cadena. La ventaja es que receptor podrá asociar eventos a etiquetas específicas .
- *agent_meet:* este comando es usado para requerir conexiones directas con un receptor específico. El receptor acepta la conexión usando el comando *agent_meet* o el comando *agent_accept*.

8.3.5 La seguridad en D'Agents

La seguridad siendo un aspecto de vital importancia en cualquier sistema de agentes móviles es considerado ampliamente en D'Agents, en el prototipo inicial se consideraban cuatro áreas principales:

- Proteger la máquina.
- Proteger a agentes de que otro agente quiera robar sus recursos.
- Proteger al agente de falsificación o robos de información.
- Proteger a un grupo de máquinas de que un agente consuma recursos excesivos en la red.

Solo los dos primeros de estos problemas son solucionados en la implementación actual de D'Agents [GRAY96] usando PGP (del inglés *pretty good privacity*) para la autenticación de los agentes y Safe Tcl para la autorización y cumplimiento de las medidas de seguridad.

8.3.5.1 Autenticación

La autenticación en D'Agents está basada en PGP. Éste encripta un fichero o mensajes de correo usando el algoritmo de llaves privadas IDEA y una llave privada escogida aleatoriamente, encripta la llave privada usando el algoritmo de llave pública RSA y la llave pública del receptor, y entonces envía la llave encriptada y el fichero al receptor.

En la actual implementación se ejecuta PGP como un proceso separado, guarda los datos a ser encriptados en un fichero, pide el proceso PGP para encriptar el fichero, y entonces transfiere en fichero al servidor destino.

Cuando un agente se registra con el servidor usando el comando *agent_begin*, la petición de registro esta firmada digitalmente usando la llave privada del propietario, encriptada usando la llave pública del servidor, y enviada al servidor. El servidor se asegura que al propietario del agente le sea permitido registrarse en su máquina y guardar la identidad autentificada del propietario del agente. Entonces la llave privada IDEA es usada como una llave de sesión para las comunicaciones posteriores entre el agente y su nuevo servidor registrado.

Cuando un agente migra usando el comando *agent_jump*, éste es firmado digitalmente con la llave privada del servidor actual y encriptada con la llave pública del servidor receptor. Esta aproximación requiere que los servidores tengan un alto grado de confianza entre ellos. El servidor receptor escoge si creer en tal identidad basada en la confianza del servidor transmisor. Si el servidor acepta al agente, guarda la identidad aparente del propietario del agente, la identidad autentificada del servidor transmisor y su grado de confianza a la que la identidad del propietario es acreedora. Una llave de sesión es usada para comunicaciones posteriores como en el caso de *agent_begin*. Los mismos pasos también ocurren cuando un agente envía un mensaje a otro agente en otra máquina.

8.3.5.2 Dos debilidades del esquema de autenticación.

La primera es que no hay un mecanismo automático de distribución para las llaves públicas. Cada servidor debe saber todas las llaves públicas posibles para que pueda autentificar a los agentes que llegan. Es necesario añadir dicho mecanismo para que cada servidor conozca las llaves cuando arranque.

Segundo, el sistema es vulnerable a ataques de réplicas en el cual el atacante duplica un agente que está llegando o un mensaje que se envía de un agente a otro (fuera de una conexión directa). Una solución hoy en día es tener un número de secuencia distinto para cada servidor.

8.3.5.3 Autorización y cumplimiento de privilegios

Una vez que la identidad del propietario del agente ha sido determinada, el sistema debe imponer restricciones de acceso al agente (autorización) y asegurar que el agente no viole esas restricciones (cumplimiento de privilegios). Los recursos son divididos en dos tipos: los recursos indirectos que sólo pueden ser accedidos a través de otro agente y los recursos incorporados (*builtin*) que son accedidos directamente a través de las primitivas del lenguaje por razones de eficiencia.

Para los recursos indirectos, el agente controla el recurso que refuerza las restricciones de acceso relevantes. Para cada mensaje de otro agente, el servidor local adjunta al mensaje cinco tuplas que contienen la identidad del propietario del agente y del servidor transmisor, una bandera que indica si el propietario podría ser autentificado y otra igual para el servidor transmisor y finalmente un nivel de confianza numérico que representa cuánta confianza el servidor local deposita en el servidor transmisor. El agente usa estas cinco tuplas con sus listas de acceso internas para responder apropiadamente a los mensajes que llegan.

Para los recursos incorporados, la seguridad es mantenida usando *Safe Tcl* y un conjunto de agentes gestores de recursos. *Safe Tcl* es una extensión de *Tcl* que es diseñada para permitir la ejecución segura de programas que no son confiables (*untrusted*). Éste provee dos interpretes, uno de los intérpretes es un intérprete confiable que tiene acceso a los comandos estándar de *Tcl*. El otro intérprete es un intérprete no confiable del cual todos los comandos peligrosos han sido removidos. Los comandos peligrosos incluyen cosas como la apertura y escritura de ficheros, creación de una conexión red, etc.

El código no confiable se ejecutará en el intérprete no confiable. Sin embargo, como no se pretende negar el acceso a dichos comandos completamente, *Safe Tcl* puede reemplazar el comando con un enlace a un comando en el intérprete seguro. La versión segura del comando verifica un conjunto de listas de acceso para ver si el comando es permitido. En la implementación actual hay una lista de acceso para el tiempo de CPU, la pantalla, la red, el sistema de ficheros, y los programas externos. Cada lista de acceso es un conjunto de pares (*nombre, cantidad*) donde el *nombre* especifica el nombre del recurso requerido y *cantidad* específica el número de instancias del recurso. El intérprete aborta un programa si dicho programa excede y los límites establecidos.

Para obtener tiempo adicional o para obtener acceso a otro recurso incorporado, el agente debe pedir permiso implícita o explícitamente a una agente gestor recursos. Hay cinco gestores de recursos en el sistema actual que corresponden al tiempo de CPU, la pantalla, la red, el sistema de ficheros, y los programas externos.

Un agente usa el comando *require* para pedir acceso explícitamente a un gestor de recursos. Este procedimiento contacta al gestor de recursos apropiado y pasa la lista de

accesos requerida al gestor de recursos. El procedimiento espera la respuesta y entonces añade cada petición de acceso a la lista de acceso interna, indicando a ambas partes si la petición fue aceptada o denegada.

Para pedir implícitamente acceso a un gestor recursos, un agente simplemente llama a un comando que use ese recurso. Por ejemplo si un agente ejecuta el comando *exec ls*, el procedimiento *exec* en el intérprete confiable verifica su permiso en la lista de acceso de programas externos. Es decir los comandos contactan al gestor de recursos del programa y continúan o se abortan dependiendo de la respuesta del gestor.

Existe un conjunto de listas de acceso para cada propietario, de tal forma que si a un agente se le ha concedido permiso para acceder a un recurso del sistema, todos los agentes del mismo propietario tendrán el mismo derecho. Es decir en la implementación actual del sistema no es posible asignarle permisos a un agente en forma particular.

De igual forma la implementación actual no provee una versión segura para todos los comandos que se consideran peligrosos. No obstante, no es posible el acceso directo a los recursos del sistema no hay forma de que un agente altere el sistema gestor de recursos ya que no hay manera que el agente modifique las listas acceso contenidas en intérprete confiable.

Adicionalmente a los gestores recursos, *Tcl* incluye un agente *consola* el cual permite al propietario de una máquina negar la entrada a un agente y permite que un gestor de seguridad pueda preguntar al propietario si un agente debería ser capaz de ejecutar una acción en particular.

8.3.5.4 Las deficiencias del esquema de autorización y cumplimiento de privilegios

Aunque en términos generales podemos considerar que el esquema de seguridad presentado es eficiente, tiene dos deficiencias evidentes: la principal es que la utilización de dos intérpretes repercuten en el tiempo de respuesta y por consiguiente en el desempeño del sistema. La segunda es la carencia de asignación de permisos de forma particular para cada agente ya que el nivel de seguridad sólo considera permisos a nivel propietario o autoridad con lo que varios usuarios que representan a una autoridad tendrían los mismos permisos de acceso. No obstante, estas deficiencias no son de vital importancia para las aplicaciones de agentes móviles.

8.3.6 Evaluación de D'Agents

Las características de D'Agents lo hacen apropiado para la experimentación con agentes en Internet y para el desarrollo de aplicaciones de tamaño medio o pequeño en las cuales al menos algún soporte de bajo nivel esta disponible en cada servidor. A continuación mostrara sus principales ventajas y desventajas.

8.3.6.1 Ventajas

- D'Agents usa un lenguaje de codificación simple *Tcl*, como lenguaje principal. Adicionalmente provee una herramienta gráfica que es de alto nivel y flexible como lo es *Tk ToolKit*. Por tanto permite un desarrollo mucho más rápido de aplicaciones de tamaño medio. Permitiendo inclusive llevar consigo una interfaz gráfica.
- Conserva el estado de ejecución perfectamente, cosa que los sistemas de agentes móviles basados en Java no pueden hacer .
- Proporciona primitivas simples y flexibles de migración y comunicación las cuales esconden los detalles de la transmisión pero son suficientemente de bajo nivel como para soportar un rango de servicios de comunicación de alto nivel.
- El sistema de seguridad es aceptable con lo que garantiza una confiable utilización de los recursos.

8.3.6.2 Desventajas

- El lenguaje que se utiliza *Tcl*, no es ampliamente conocido y sólo trabaja en equipos UNIX.
- El uso de dos intérpretes hace que sea muy lento en comparación con otros lenguajes de codificación, y mucho más lento que los intérpretes de *bytecode*.
- No proporciona una modularización de código, no hay jerarquías de clases ni lugares (*places)* lo que limita la estructuración de código y lo hace inapropiado para grandes aplicaciones.
- Las primitivas de comunicación tienen dos desventajas. La primera es que requiere un protocolo de comunicación de alto nivel, este deber ser implementado sobre de las primitivas de bajo nivel. La segunda es que no hay un lenguaje común que todos los agentes entiendan.
- La principal debilidad es que no tiene características avanzadas de desarrollo, como por ejemplo un entorno visual de depuración.

8.4 ARA: AGENTES PARA ACCESO REMOTO

Ara es un proyecto del área de sistemas distribuidos del departamento de informática de la Universidad de Kaiserslautern [KAISa], Alemania. La idea básica del sistema de agentes para acceso remoto ARA (de la lengua inglesa *Agents for Remote Access)* es crear una plataforma capaz de moverse libre y fácilmente sin interferir con su ejecución, utilizando diferentes lenguajes de programación existentes, independiente de los sistemas operativos y de las máquinas que participan. Completando esto, el sistema proporciona facilidades para los requerimientos específicos de los agentes móviles en aplicaciones reales, en donde la seguridad viene a ser el más prominente.

Ara es un buen ejemplo de software intermedio, situado entre las aplicaciones específicas y el sistema operativo que hay debajo. Proporciona las facilidades al nivel de sistema para ejecutar y mover programas, permitiéndoles interactuar y acceder a su sistema servidor, todo ello de una forma portable y segura .

8.4.1 Generalidades de Ara

Ara consiste en un grupo de agentes moviéndose entre lugares, donde ellos usan ciertos servicios para hacer su trabajo. Como los agentes son escritos en algunos lenguajes de programación, el papel de los lenguajes dentro del sistema es de fundamental interés [COCK98].

8.4.1.1 Agentes, lenguajes, y el sistema Ara

Los agentes móviles en Ara son programados en un lenguaje interpretado y ejecutados dentro de un intérprete para este el lenguaje, usando un *runtime* especial para los agentes, llamado el núcleo en términos de Ara. Sin embargo la relación entre el núcleo y el intérprete es característico de Ara: aislar los temas específicos del lenguaje en el intérprete, mientras se concentra toda la funcionalidad independiente del lenguaje en el núcleo.

Esta separación de intereses hace posible utilizar diversos intérpretes a la vez para diferentes lenguajes de programación sobre un núcleo común y genérico. El núcleo trabaja sólo con los agentes en general, haciendo sus servicios disponibles uniformemente a todos los agentes sin importar sus respectivos lenguajes interpretados. El esquema completo de Ara se simplifica en: intérpretes y núcleo corriendo como un proceso de aplicación simple sobre un sistema operativo servidor sin modificaciones.

Actualmente, los intérpretes para los lenguajes programación *Tcl* y *C* han sido adaptados en el núcleo de Ara, y Java será añadido próximamente.

Cuando se requiere la completa funcionalidad de un agente móvil, los agentes compilados son integrados en el sistema como una alternativa eficiente para casos en donde ciertos requerimientos de seguridad y portabilidad pueden ser sacrificados. Los agentes compilados son empleados usualmente para servicios residentes en el sistema local .

Así con Ara se tiene la opción de escoger un lenguaje de programación, en vez de requerir que todas las aplicaciones estén escritas en un mismo lenguaje preestablecido.

8.4.1.2 La vida de un agente

La creación y el borrado de un agente son funciones básicas ofrecidas por el núcleo. A cada agente le es asignado un identificador único, el cual no puede ser modificado. En Ara los agentes pueden reproducirse, suspender y reanudar temporalmente su ejecución, dormir por un tiempo e inclusive terminar su ejecución.

Estas funciones de control de agentes pueden ser usadas para formar un equipo de trabajo sobre una tarea común. Los agentes pueden hablar uno con el otro, intercambiar información, ofrecer y requerir servicios entre ellos e inclusive negociarlos.

Ara atendiendo a la definición de agente móvil sólo permite la interacción local entre agentes, existen varias opciones para el esquema de interacción, incluyendo ficheros de

disco, áreas compartidas de memoria, intercambio de mensajes directamente y llamadas a procedimientos especiales. El núcleo proporciona un punto de servicio para esta interacción llamado punto de encuentro (*meeting place)* en donde los agentes interaccionan como clientes o como servidores.

El servidor por otra parte establece límites para los agentes visitantes que se ejecutan en el sistema. Los agentes en Ara están equipados con cuentas de recursos llamadas asignaciones (*allowances)* las cuales registran la cantidad que un agente puede consumir de ese recurso en el futuro, cuando el agente consume algo de ese recurso su respectiva asignación se actualiza.

El sistema asegura que un agente nunca sobrepase su asignación. Un grupo de agentes puede también compartir una asignación común, cada consumición de esta será actualizada de acuerdo a su propia política. Los agentes pueden indagar acerca del estado actual de su asignación en cualquier momento e inclusive pueden transmitirlas de uno a otro.

8.4.1.3 Movilidad de agentes

Los agentes en Ara pueden migrar en cualquier punto de su ejecución, simplemente al usar una llamada especial al núcleo, llamado *ara_go* en la interfaz de *Tcl* de Ara.

La instrucción *ara_go* es todo lo que el programador necesita saber acerca de la migración. El sistema asegura que el agente es transferido completamente al lugar indicado y lo reanuda exactamente donde esté se interrumpió, es decir, exactamente después de la instrucción *ara_go*.

Los agentes de pueden moverse entre lugares, los cuales son una asociación lógica a una localización física y son un concepto en arquitectura de Ara. Los lugares son localizaciones virtuales dentro del sistema, los cuales se están ejecutando sobre una cierta máquina. Adicionalmente a la estructuración que proveen, los lugares facilitan el control sobre los agentes que admiten. Los lugares juegan un papel central en la política de seguridad del sistema de Ara. De hecho, cada lugar puede implementar su propia política de seguridad.

8.4.1.4 Acceso al sistema servidor

Los agentes de Ara tendrán acceso a un sistema de ficheros jerárquico. La principal mejora, además de esconder las dependencias de las plataformas, será el control de acceso. Los ficheros pueden ser incluidos por sus propietarios en listas de control de acceso, permitiendo o denegando selectivamente el acceso a agentes específicos o a grupos de agentes.

Como algunos agentes comparten recursos, bloqueos para controlar la concurrencia de lectura escritura serán proveídos por el sistema.

Las capacidades para el acceso al sistema de ficheros son considerados como medios adicionales de autorización. Estas son más que flexibles, pero más difíciles de gestionar.

Las capacidades pueden restringir individualmente por límites de volumen o por el número de veces de validez .

El acceso a las interfaces gráficas de usuario estará basado en *Tk*. Éste será mejorado significativamente para hacerlo compatible con algunas características de Ara. Las interfaces de agentes clientes para aplicaciones externas en la máquina local serán usualmente implementados por agentes *proxy*, los cuales son agentes estacionarios confiables dedicados a representar las aplicaciones externas para el sistema de agentes.

Un concepto general de canales de comunicación será proporcionado, para el acceso de la red y la comunicación local con entidades.

Actualmente, como la interfaz del servidor no está disponible, los agentes ejecutan operaciones de entrada/salida usando las facilidades estándar de su lenguaje respectivo.

8.4.2 La arquitectura de Ara

Los agentes se ejecutan dentro de interpretes para su respectivos lenguajes de programación, controlados y servidos por el núcleo de un sistema común. Éste es el principio fundamental de Ara ejecutar los agentes como procesos concurrentes y autónomos. Esta idea soporta su independencia y posiblemente la ejecución asíncrona, provee control flexible y facilita la protección mutua.

8.4.2.1 Procesos y arquitectura interna

El núcleo de Ara proporciona su propia abstracción de procesos como la base para la implementación de agentes. Existe usualmente un intérprete ejecutándose, el cual procesa el programa de un agente móvil. El núcleo gobierna los procesos del agente, y realiza la mediación su interacción y el acceso al servidor del sistema.

Las funciones básicas, como migración, son proporcionadas a los agentes por el núcleo, mientras que los servicios de alto nivel son ofrecidos por agentes servidores. Los agentes estacionarios pueden ser compilados ya que éstos serán locales y contribuirán con el núcleo en la gestión del sistema .

El sistema de Ara también emplea procesos para ciertos propósitos internos, con el fin de modularizar la arquitectura. Los procesos del sistema tienen permisos especiales y pueden directamente acceder al sistema operativo servidor, pasando por el núcleo. Como la implementación de procesos es interna en el sistema de Ara, el ensamblaje completo de agentes, intérpretes y núcleo se ejecutará como un proceso simple de aplicación en la cima de un sistema operativo servidor intacto, como se puede apreciar en la figura 8.4.2.1. Lo que facilita considerablemente la portabilidad a plataformas específicas.

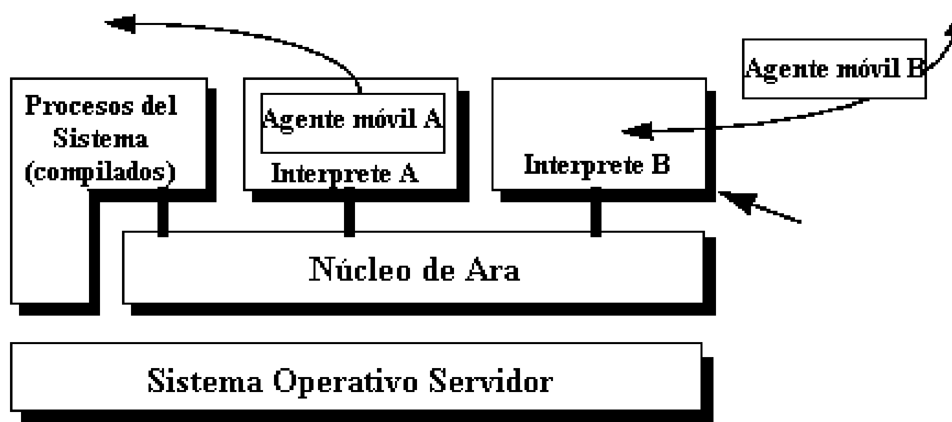


Figura 8.4.2.1 - Vista de alto nivel de la arquitectura del sistema de Ara

Los procesos en Ara son implementados como hilos individuales de control. El núcleo contiene un paquete eficiente de hilos, el cual los ejecuta concurrentemente y sin derecho a prioridades. El espacio común de direcciones permite una alta eficiencia en la gestión de procesos y en la interacción. Gracias a la planeación de procesos, no hay necesidad para la sincronización en el núcleo ni en los intérpretes, beneficiando así el desempeño.

8.4.2.2 El proceso de comunicación

El proceso de comunicación es el más importante del sistema, gestiona la interfaz de red del servidor con el propósito de la migración de agentes. Éste acepta agentes que han sido enviados en la forma de vectores linealizados y también los envía al proceso de comunicación del servidor destino.

El proceso de comunicación puede gestionar cualquier número de agentes (que llegan a y salen del servidor) en paralelo intercalando las transmisiones individuales. Los agentes son transmitidos como un paquete de datos unidireccional del proceso de comunicación del servidor origen al del servidor destino.

8.4.2.3 Protección

La protección en un contexto general se refiere al control de los datos leídos y escritos así como las llamadas a funciones externas por parte del agente. En relación a las llamadas de funciones, la protección es trivial ya que cada llamada tiene que pasar a través de un *stub* definido por el intérprete el cual puede ser confiables debido a que está fuera del alcance del agente.

El intérprete asegura que un agente sólo tiene acceso directo a su propio espacio de direcciones, mientras las funciones del núcleo deben ser usadas para interaccionar con otros agentes y con el núcleo. En núcleo y sus objetos, tales como los otros agentes y los puntos de servicio, existen fuera del espacio de los agentes en la memoria real.

Cuando una función del núcleo es llamada por un agente, los objetos que se pasan como argumentos son proporcionados en la que forma de identificadores los cuales se hacen corresponder con sus objetos por el *stub* para esta función usando una tabla de objetos mantenida por el núcleo. Los *stubs* también ejecutan cualquier verificación de parámetros específicos del lenguaje.

8.4.2.4 Salvado y restauración del estado de un proceso

Los agentes de Ara pueden moverse sin interferir con su ejecución. Esta migración implica que el estado de ejecución tiene que ser extraído del sistema origen y reinstalado en el sistema destino. Algunas ocasiones esto se convierte en un proceso delicado, sobre todo cuando los servidores participantes tienen arquitecturas diferentes.

En Ara el estado de un agente es transformado a una forma portable antes de ser movido. El estado de ejecución de un agente está compuesto por el estado de su intérprete específico por un lado y por el estado de su proceso general en el núcleo de Ara sobre el que reside .

El núcleo ejecuta la transformación portable de la parte de Ara por sí mismo, mientras que usa una función dedicada definida por el intérprete (*upcall)* para la otra parte. Esta función puede construir su implementación en un número de funciones de utilidad ofrecidas por el núcleo. Ésta proporciona la transformación de tipos del datos comunes a una forma portable.

8.4.2.5 Relación entre el núcleo y los intérpretes

El intérprete en sí es conceptualmente una parte del sistema de Ara, es confiable, y admite al núcleo en sus tareas quien puede penetrar al agente interpretado. Los deberes de un intérprete incluyen la definición de interfaces (*stubs)* [KAISb] que son interfaces de llamada en sus lenguajes de programación para invocar las funciones de la API proporcionadas por el núcleo e inversamente proveen funciones para la gestión del intérprete (*upcalls)* al núcleo. El trabajo de los *stubs* es principalmente en la conversión de formatos de datos y en la traducción de interfaces similares, en particular, ellos deben hacer corresponder los identificadores para los objetos del núcleo y los objetos mismos [PEIN97a]. Obsérvese la figura 8.4.2.5.

Un requerimiento general para el intérprete es satisfacer las demandas de memoria y dinámica durante la interpretación. Más aún, el intérprete debe proveer una imagen de memoria virtual.

Como vimos anteriormente, cualquier lenguaje de programación interpretado puede ser adaptado a Ara: cualquier intérprete dado debe ser extendido por mecanismos que permitan servir y proporcionar las funciones descritas.

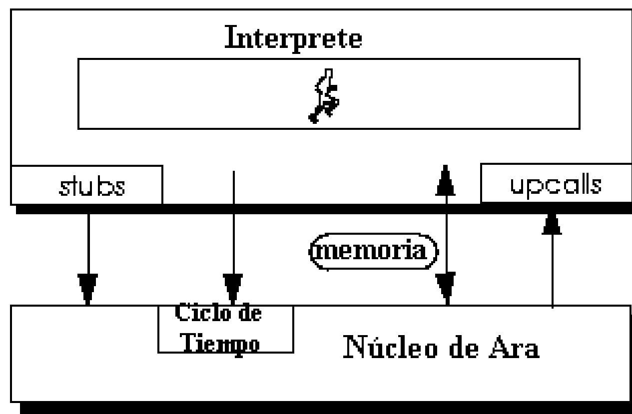


Figura 8.4.2.2 - Adaptación del interprete de un lenguaje al núcleo de Ara

8.4.3 Características de Ara y conceptos de programación

Un los agentes en Ara acceden a la funcionalidad del sistema llamando a las interfaces de aplicación del núcleo. Técnicamente, estas funciones están compiladas en el código nativo que se encuentra contenido en cada sistema Ara. Cada una es accesible por un agente través de una interfaz de llamadas (*llamada stub)* en el lenguaje respectivo, el cual es parte del lenguaje interpretado. Los diversos *stubs* para el mismo núcleo difieren en sintaxis, pero no en semántica.

Los *stubs* definidos para el uso de los agentes en 'C' son usualmente nombrados que en las funciones en el núcleo mientras que las funciones en *Tcl* tienen sus propios nombres. Para evitar la duplicidad de funciones al referirme a cada uno de los lenguajes mencionaré el nombre de las funciones que tienen en el núcleo.

8.4.3.1 Comandos para la gestión básica de un agente

- *ara_agent*: es la función que permite crear un agente para cada implementación del lenguaje, pasándole un código así como parámetros de inicialización.
- *ara_agents*: proporciona el conjunto de agentes que se encuentran en ese momento activos en el sistema.
- *ara_exit*: permite la terminación voluntaria del agente en cualquier momento.
- *ara_kill*: elimina a un agente siempre y cuando se tengan los privilegios necesarios.
- *ara_shutdown*: termina el proceso raíz y por consiguiente todo el sistema. Este comando puede ejecutarse en la línea de comandos.
- *ara_suspend*: suspende al agente y lo envía a su estado de esperar.
- *ara_active*: activa un agente lo que significa que lo lleva del estado de esperar al estado de listo.
- *ara_retire*: libera temporalmente el control de ejecución para que otro agente con su hilo correspondiente lo aproveche.

8.4.3.2 Comandos relacionados con el tiempo

- *ara_now*: devuelve el tiempo actual en el reloj lógico del sistema como un entero positivo.
- *ara_sleep*: suspende a un agente por un cierto período de tiempo lógico.

8.4.3.3 El comando de movilidad.

- *ara_go*: permite a un agente migrar al lugar destino especificado, llevando con en sus asignaciones.

De considerarse que los agentes pueden tener asignaciones locales que sólo son válidas para el lugar actual. Existe una restricción importante para la operación de migración: el estado externo de un agente (sus relaciones a otros sistemas de objetos y recursos como ficheros y ventanas) no migran con él, desde que en estos objetos dependen de la máquina local.

8.4.3.4 Comandos que definen el comportamiento de lugares.

- *ara_placeNameCreate*: crea un nuevo lugar en la máquina local.
- *ara_here*: permite a un agente encontrar el nombre de su lugar actual en la máquina local.
- *ara_placeNameEqual*: permite comparar si el nombre de dos lugares son iguales.
- *ara_placeNameDelete*: elimina el lugar que se especifica del sistema.

8.4.3.5 Comandos para la gestión de asignaciones

- *ara_get_allowance*: a través de este comando un agente puede averiguar su propia asignación local y local en cualquier momento.
- *ara_transfer_allowance*: este comando permite transferir una parte de la asignación que tiene un agente origen a la asignación que tiene un agente destino .
- *ara_groupSize*: permite averiguar el número de miembros de su propio grupo.

8.4.4 La seguridad en Ara

La capa más básica de seguridad en la arquitectura de Ara es la protección de memoria a través de los intérpretes. Además de esta protección fundamental, los diferentes lugares que existen en el sistema de Ara juegan un papel central en el concepto de seguridad. Un lugar en Ara establece un dominio de servicios lógicamente relacionados bajo una política de seguridad común que gobierna a todos los agentes de ese lugar [PEIN97b].

8.4.4.1 Asignaciones para limitar el acceso a recursos.

La función central de un lugar es decidir las condiciones de admisión, si de alguna manera, un agente trata de entrar. Estas condiciones son expresadas en la forma de una asignación, la cual es concedida al agente por el tiempo que éste permanecerá en ese lugar. Una asignación es un vector de permisos de acceso para varios recursos del sistema, como ficheros, tiempo de CPU, memoria, etcétera. Los límites pueden ser cuantitativos o cualitativos dependiendo del recurso de que se trate. Un agente que migra hacia un lugar específica la asignación que él desea tener para realizar sus tareas, el lugar en turno decide que asignación le concederá[18] al aplicante y se la impondrá cuando el agente entre. El núcleo del sistema en Ara asegurará que ningún agente sobrepasará su asignación. .

Además de la asignación local, cada agente podrá ser equipado con una asignación global en el tiempo de su creación. La asignación global pone límites sobre todas las acciones de un agente durante de su ciclo de vida, limitando efectivamente los recursos de acuerdo a sus objetivos. El núcleo del sistema nunca concederá que una asignación local exceda la asignación global del agente. Los agentes podrán averiguar acerca del estado de sus asignaciones (global y local) en cualquier momento. Y podrá transferir cantidades de ella a otro agente bajo ciertas condiciones. Los agentes pueden también formar grupos para compartir una asignación común.

8.4.4.2 Entrada a un lugar

Los lugares pueden ser creados dinámicamente, especificando un nombre y una función de admisión. Esta función tiene una interfaz definida, recibe el nombre del agente y el grado de confianza de la autenticación con la asignación local deseada como parámetros de entrada, la función de admisión regresa la asignación local que será impuesta sobre este agente o bien la denegación de su admisión. Cada lugar por lo tanto debe implementar su propia política de seguridad, discriminando a agentes individuales, autoridades, o dominios de origen; controlando así el acceso a los recursos con la granularidad apropiada.

A un agente que le ha sido denegado el acceso al lugar destino es regresado a su lugar de origen para que descubra la falla o razón por la que no pudo acceder.

Las restricciones de acceso a recursos generales impuestas por las asignaciones son un mecanismo de seguridad adecuado para los accesos comunes. Sin embargo, ciertos requerimientos de seguridad de alto nivel como el obligar que sólo datos de un formato específico sean enviados, requieren correspondientemente restricciones de acceso de alto nivel. Esto puede ser logrado usando puntos de servicio como salidas controladas del dominio de seguridad, proporcionadas por un agente confiable quien mantiene todos aquellos requerimientos de alto nivel.

[18] La decisión la tomará en base a la autenticación. Sin embargo esta asignación que representa una parte fundamental del sistema de seguridad de Ara esta aún en construcción.

8.4.4.3 Problemas abiertos.

Al igual que la mayor parte del sistema de seguridad de Ara. La parte de autenticación está siendo implementada y estará basada en firmas digitales usando criptografía de llave pública. Sin embargo como el agente móvil usualmente cambia durante su itinerario, no puede ser firmado en todos sitios por su autoridad, lo cual hace difícil la autenticación de las partes cambiantes del agente.

Otros componentes de seguridad relevantes de un agente móvil podrían ser autentificados por esquemas dedicados. Adicionalmente a la autenticación, la criptografía de llaves públicas será también usada para encriptar a los agentes de Ara durante la migración para protegerlos de espionaje. Como en cualquier esquema criptográfico, la pregunta de distribución de llaves debe ser resuelta. Ara no tiene ningún soporte para esto.

También existe el problema inverso de la seguridad de los agentes contra acciones indeseadas del servidor como por ejemplo espiar el contenido del agente o modificarlo para causarle un efecto nocivo. El sistema de Ara proporcionará ciertas medidas como la protección de partes inmutables del agente, sin embargo, cosas como el espionaje del contenido del agente no pueden ser resueltas por medios técnicos.

8.4.5 Evaluación de Ara

Sus características lo hacen ideal para los sistemas de agentes que requieren ser creados en distintos lenguajes. Las facilidad de que los agentes puedan ser creados en 'C', Tcl, y próximamente en Java ofrece una virtud única de interacción de distintos agentes programados en diversos lenguajes sobre un mismo núcleo y soportados por una sola plataforma.

8.4.5.1 Ventajas

- *Lenguajes múltiples*: como ya lo hemos mencionado el sistema ofrece diversos lenguajes para la creación de agentes, y está preparado para añadir más lenguajes a la plataforma, basta con añadir el intérprete apropiado para el lenguaje deseado.
- Su esquema de seguridad permite restringir los recursos que consumirá un agente durante su ciclo de vida por medio de su asignación global .
- La separación de los intérpretes y el núcleo ha sido bien diseñada, de tal forma que mediante el uso de *stubs* en los intérpretes se accede a las funciones nativas en el núcleo sin afectar el rendimiento del sistema.
- La existencia de lugares proporciona la posibilidad de modularizar un sistema complejo así como de que añadir políticas de seguridad propias para cada lugar.
- Es más estructurado y rápido que D'Agents.

8.4.5.2 Desventajas

- El esquema de seguridad de autenticación no ha sido desarrollado aún, por lo que se confía de 'buena fe' en la identidad de los agentes que llegan. Es necesario que dicho esquema sea añadido para que la plataforma pueda ser utilizada en proyectos reales. Adicionalmente se requiere la encriptación de los agentes al ser transferidos.
- En una red de tamaño real, los agentes móviles no pueden saber los lugares de interés para desarrollar sus tareas. Se requiere que un servicio de directorio (páginas amarillas) se ha añadido para éste haga corresponder los servicios con los lugares.
- Carece de más protocolos de comunicación además de TCP para realizar la migración.
- Hasta ahora sólo es soportado sobre sistemas operativos UNIX. Es necesario que sea soportado bajo los sistemas operativos Windows.
- Los lenguajes con los que se pueden crear agentes carecen de todas las ventajas de la orientación a objetos, con lo que la creación de sistemas grandes se ve menguada.

8.5 IBM AGLETS

Los aglets son objetos Java que se pueden mover a través de la red de un ordenador a otro. Es decir, un aglet que se este ejecutando en una máquina puede detener su ejecución, desplazarse a otra máquina remota y reiniciar su ejecución de nuevo en dicha máquina. Cuando un aglet se mueve, lleva consigo el código del propio aglet así como el estado de todos los objetos que lo constituyen. Un mecanismo de seguridad interno hace que sea seguro el hospedar aglets no confiables (untrusted).

El sistema tiene los siguientes objetivos:

- Proveer un modelo de fácil compresión para la programación de agentes móviles sin tener que realizar modificaciones en la máquina virtual de Java JVM o código nativo.
- Soportar una comunicación dinámica y potente que permita a los agentes comunicarse con otros agentes tanto conocidos como desconocidos.
- Diseñar una arquitectura reusable y extensible.
- Diseñar una arquitectura que sea compatible con la tecnología Web/Java existente.
- Proveer de mecanismos de seguridad que sean de fácil compresión y lo suficientemente simples para permitir que los usuarios finales puedan especificar una serie de restricciones de acceso a los agentes móviles.

8.5.1 El modelo del objeto Aglet

Los aglets no son más que objetos Java con capacidad de migración, las clases[19] heredan su funcionalidad de un conjunto de interfaces que definen su

[19] Que al ser instanciadas se convertirán en un agente móvil

comportamiento[OSHI97]. Presentaré con más detalle cada uno de los elementos de este modelo de objetos.

8.5.1.1 Generalidades de la API de los aglets

El Aglet API define la funcionalidad principal de los agentes móviles. La siguiente figura muestra las interfaces y las clases principales definidas en la API de los Aglets, así como la relación existente entre las interfaces

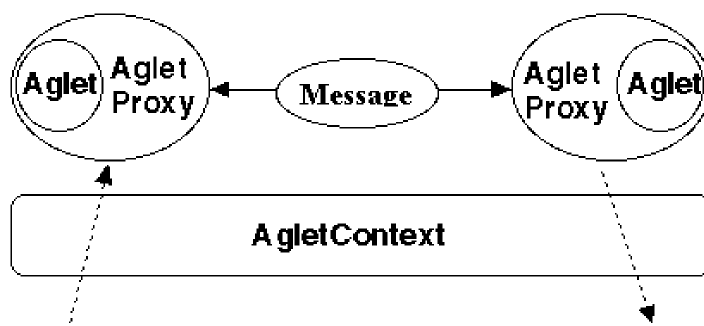


Figura 8.5.1.1 - Interfaces y clases principales de la API de los Aglets

- *aglet.Aglet:* define los métodos fundamentales (por ejemplo, dipatch(URL)) que permiten a un agente móvil controlar su movilidad y ciclo de vida. Todos los agentes móviles definidos en Aglets tienen que heredar de esta clase abstracta. Las principales primitivas de aglet.Aglet son:
  - Aglet.dispatch(URL) permite que un aglet se mueva desde una máquina local a un destino que se le pasa como argumento.
  - Aglet.deactivate(long time) permite que un aglet se almacene en memoria secundaria.
  - Aglet.clone() crea una nueva instancia de un aglet que posee el estado del aglet original. Cabe destacar que el objeto devuelto por clone() no es un objeto Aglet sino un objeto AgletProxy.

  La clase Aglet se usa también para acceder a los atributos asociados a un aglet. El objeto aglet.AgletInfo, que puede obtenerse haciendo uso del método Aglet.getAgletInfo(), este objeto contiene los atributos inherentes al aglet, tales como su hora de creación y su código base, así como sus atributos dinámicos, entre los cuales se encuentran la hora de llegada y la dirección del contexto actual.

- *aglet.AgletProxy:* la interfaz del AgletProxy actúa como un gestor del aglet y ofrece una forma de acceder al aglet que se encuentra tras él. Puesto que un aglet posee varios métodos públicos que no deben accederse directamente desde otros aglets por motivos de seguridad, cualquier aglet que desee comunicarse con otro aglet tiene que obtener primeramente el objeto proxy, para así poder interactuar con el aglet a través del interfaz que ofrece el proxy, es decir el proxy del aglet se comporta como un escudo que protege al aglet de agentes malignos. Una vez invocado el objeto proxy consulta al SecurityManager para determinar si en el contexto actual de ejecución se pueden ejecutar métodos. Otra función importante del AgletProxy es darle transparencia a la ubicación del aglet. Si un aglet determinado reside en una máquina remota, redirecciona las solicitudes a dicha máquina y devuelve el resultado a la máquina local. El objeto AgletProxy puede obtenerse de las siguientes formas:

- Obtener una enumeración de los proxies en un contexto llamando a la primitiva AgletContext.getAgletProxies().
- Obtener un AgletProxy para un identificador de aglet dado a través de: AgletContext.getAgletProxy(AgletID*)* o bien de Aglets.getAgletProxy(String NombreContexto, AgletID).
- Obtener el objeto AgletProxy por paso de mensajes. Un objeto AgletProxy puede pasarse como argumento en un objeto *Message* y ser enviados a un aglet local o remoto.
- Introducir un objeto AgletProxy en el ContextProperty (objeto que define las propiedades del contexto)
  para hacer esto debe llamarse al método AgletContext.setProperty (String,Object), que permitirá compartir el objeto proxy.

La librería runtime es responsable de la interfaz AgletProxy, así los programadores de aglets no tendrán que implementar esta interfaz.

- *aglet.AgletContext:* ofrece una interfaz para el entorno de ejecución que ocupa el aglet. Cualquier aglet puede obtener una referencia a su objeto AgletContext a través de la primitiva Aglet.getAgletContext().aglet.Message, este objeto puede usarse para obtener información local como por ejemplo la dirección del contexto que alberga al aglet o una enumeración de objetos AgletProxy, este método también se usa para obtener el contexto y así poder crear un nuevo aglet en el contexto. Cuando un aglet se traslada a otro contexto, abandona el contexto actual y pasa a formar parte del nuevo contexto. La librería runtime es la responsable de implementar esta interfaz. Por tanto no será necesario que los programadores de aglets implementen esta interfaz.

- *aglet.message:* los objetos aglets se comunican entre sí intercambiando objetos de la clase *Message*. Un objeto *message* puede tener un objeto String para indicar el tipo de mensaje. También puede tener objetos arbitrarios como argumentos. Un mensaje se envía a un aglet haciendo uso de alguno de los siguientes métodos: AgletProxy. sendMessage(Message msg), FutureReply AgletProxy.sendAsyncMessage(Message msg), void AgletProxy.sendOnewayMessage(Message msg). El mensaje se pasa como argumento a Aglet.handleMessage(Message msg). Véase la sección sobre mensajes para más detalles.

- *aglet.FutureReply:* El objeto definido por la interfaz FutureReply se devuelve en el paso de mensajes asíncrono y se usa como un gestor para poder recibir la respuesta de mensajes de forma asíncrona. Con esta interfaz, se puede determinar si hay una respuesta disponible, y se puede esperar por la respuesta durante un periodo de tiempo determinado, pudiendo continuar el aglet su ejecución si no se ha recibido una respuesta una vez transcurrido el intervalo de tiempo especificado. En las siguientes secciones se verán más clases e interfaces que permiten controlar las actividades de un aglet.

8.5.1.2 Heredando de la clase aglet

Las reglas generales para heredar de la clase Aglet son las siguientes:

- Una clase Aglet definida por el usuario normalmente deberá heredar de la clase aglet.Aglet.
- Se puede heredar de cualquier otra implementación de la clase Aglet.
- Puede implementarse una interfaz java.io.Externalizable.
- No se podrán sobrescribir métodos del Aglet API, tales como dispatch() y getAgletContext() ya que están declarados como *final*[20].
- La clase aglet no debería implementar un constructor sin parámetros. No obstante es posible que se tenga que implementar un constructor vacío si el aglet implementa java.io.Externalizable. ObjectSerialization invoca al constructor del objeto que implementa una interfaz Externalizable, y muestra un error si no existe un constructor público. Nótese que el constructor debe estar vacío, puesto que al constructor se le llama incluso durante la deserialización de los objetos. Esto ocasiona que el objeto sea inicializado cada vez que el objeto se despacha o se desactiva. Todo esto no ocurre a menos que el aglet implemente java.io.Externalizable.

Los siguientes métodos de la clase Aglet se supone que serán sobrescritos por una subclase para permitir que un aglet implemente un comportamiento especifico.

- *void AgletOnCreation(Object init)*: este método se va a llamar sólo una vez durante el ciclo de vida del aglet, y es precisamente cuando se crea el aglet.
- *void Aglet.onDisposing():* este método se llama cuando se deshecha un aglet. El aglet debe devolver todos los recursos utilizados. También podrá realizar ciertas acciones antes de ser eliminado.
- *void Aglet.run():*el método run() de la clase Aglet se llama cuando una instancia es reconstruida, es decir, cuando se crea una instancia, cuando es clonada o cuando es activada. Puesto que el método se llama cuando el aglet ocupa el contexto, este es un buen lugar para definir una tarea común
- *Boolean Aglet.handleMessage(Message msg)*: Todos los mensajes que se envían a un aglet son tratados por el método handleMessage. Los programadores de aglets pueden comprobar si el mensaje recibido es un mensaje conocido para así poder realizar la tarea que se corresponda con el tipo del mensaje.

Obsérvese el siguiente ejemplo en donde se aprecia el uso de algunos de los métodos descritos:

```
public class HelloAglet extends Aglet {
      public void onCreation(Object init) {
         System.out.println("¡Creado!");
      }
      public void run() {
         System.out.println("¡Hola!");
      }
      public boolean handleMessage(Message msg) {
         if (msg.sameKind("DecirHolaOtraVez") {
            System.out.println("¡hola!");
            return true;
         }
         return false;
```

[20] Palabra reservada de Java que indica que un método no se puede sobrescribir.

```
    }
    public void onDisposing() {
       System.out.println("¡Adios!");
    }
}
```

8.5.1.3 El objeto aglet y su ciclo de vida

La clase Aglet facilita la funcionalidad básica de un agente móvil, por lo que todos los objetos móviles (objetos aglet) deben ser una instancia de una subclase de la clase aglet.Aglet. Para que un aglet sea funcional debe ser primeramente instanciado. Existen dos formas de crear una nueva instancia de un aglet.

La primera es instanciar un aglet completamente nuevo a partir de las definiciones de una clase, haciendo uso del método AgletContext.createAglet(URL codebase, String name, Object init). Esta primitiva crea una nueva instancia dentro del contexto especificado. También puede inicializarse el aglet si es necesario, una vez hecho esto se llamará al método Aglet.onCreation (Object.init) del objeto creado pasándole también como parámetro el objeto inicializador que se le pasó a la primitiva createAglet.

El otro camino para engendrar un aglet es crear una copia de un aglet ya existente, usando la primitiva Aglet.clone(). El aglet clonado posee el mismo estado que el aglet original, no obstante tiene un objeto AgletID diferente y por tanto una identidad unívoca.

Una vez creado, un objeto aglet puede ser enviado a una maquina remota o traído de ésta, ser a continuación desactivado y almacenado en memoria secundaria, para ser activado posteriormente.

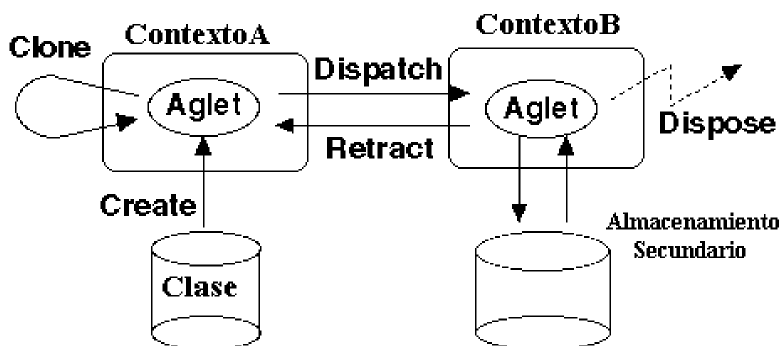


Figura 8.5.1.2 - Ciclo de vida de un Aglet

Un aglet puede enviarse a una máquina remota llamando al método Aglet.dispatch(URL dest). Para ser más preciso, un aglet ocupa un contexto y puede moverse desde este contexto a otros contextos durante su ejecución. Puesto que el sistema runtime puede servir a varios contextos dentro de un Java VM, estos contextos pueden estar en el mismo VM.

Al enviarse un aglet hace que se suspenda su ejecución, serialice su estado interno y su código binario (bytecode) a una forma estándar y a continuación sea transportado a su destino. En el extremo del receptor, el objeto Java se reconstruye de acuerdo con los datos recibidos desde el origen, una vez hecho esto se asigna un nuevo hilo y se reanuda su ejecución.

Los aglet soportan la persistencia de los objetos aglet. Todos los objetos aglet pueden ser persistentes por naturaleza, puesto que debe ser posible que se conviertan en un flujo de bits, pudiendo así este flujo almacenarse en memoria secundaria. La primitiva Aglet.deactivate(long timeout) permite que un aglet sea almacenado en memoria secundaria donde permanecerá dormido durante un número especificado de milisegundos. Una vez que dicho tiempo ha transcurrido o que otro programa ha solicitado su activación, el aglet será reactivado dentro del mismo contexto donde fue desactivado.

A diferencia de los objetos normales en Java, que son desechados normalmente por el recolector de basura, un objeto aglet, puesto que es activo, puede decidir si desea morir o no. Si se llama al método dispose()[21] para matar al aglet, el método onDisposing() se llamará a continuación para realizar las tareas de finalización acordes con el estado actual del aglet. Los programadores de aglets son los responsables de devolver los recursos utilizados por los aglets, tales como descriptores de archivos o conexiones a bases de datos, ya que estos recursos puede que no sean devueltos automáticamente.

8.5.1.4 Los eventos de los aglets y el modelo de delegación de eventos

El Java VM no permite trasladar la pila de un objeto hilo (Thread). Es imposible hacer migrar un objeto hilo a un servidor remoto o salvarlo a memoria secundaria al mismo tiempo que se guarda su estado de ejecución[22]. En lugar de esto, los aglets usan un modelo de eventos para posibilitar a los programadores la implementación de una acción que permita su movilidad.

Antes de que un aglet se envíe a si mismo se llama al método **onDispatching()** del objeto **MyListener** y el método **onArrival()** se llama antes de que el aglet llegue a su destino. De esta forma, un programador de aglets puede implementar una acción que se realizará como respuesta a estos eventos (a enviarse, por ejemplo) independientemente de cuando se envió el aglet o de quien fue el encargado de enviarlo.

Cuando	Evento	Listener	Método llamado
A punto de crearse	CloneEvent	CloneListener	OnCloning
Cuando se crea	CloneEvent	CloneListener	OnClone
Después de que el clon ha sido creado	CloneEvent	CloneListener	OnCloned
A punto de despacharse	MobilityEvent	MobilityListener	OnDispatching
Cuando se solicita su vuelta	MobilityEvent	MobilityListener	OnReverting

[21] Una vez desechado un aglet, su instancia es inservible y se convierte en una entidad inválida, cualquier intento de operar con ella lanzará una excepción de seguridad.
[22] Esto se debe a que la JVM no permite accesos a la pila.

y está a punto de retornar.			
Después de haber llegado al destino	MobilityEvent	MobilityListener	OnArrival
Después de desactivarse	PersistencyEvent	PersistencyListener	OnDeactivating
Después de activarse	PersistencyEvent	PersistencyListener	onActivation

Los objetos listener pueden conectarse y desconectarse y por tanto pueden añadirse a un aglet y borrarse de él en tiempo de ejecución, es por ello por lo que las clases listener pueden usarse como una librería de aglets. Por ejemplo puede usarse la clase **LimitadorClones** para que cualquier aglet pueda especificar su máximo número de clones.

8.5.2 Migración de objetos y mensajes con aglets

Cuando un aglet se envía, los objetos listeners correspondientes se llamarán inmediatamente. Si estas llamadas se completan con éxito, en primer lugar se suspende el hilo de ejecución, a continuación se llama a **ObjectSerialization** serializando el aglet y todos sus objetos asociados. Después de que estos objetos hayan sido serializados y transportados con éxito a su destino, se mata al hilo de ejecución y todos los recursos relacionados se devuelven al sistema.

Si un aglet ha creado más de un hilo, también se terminarán esos hilos. Para evitar cualquier problema ocasionado por una finalización inesperada, un aglet debe implementar un interfaz de un listener y terminar estos hilos en el método de llamada al listener.

8.5.2.1 Diversos tipos de serialización

- *Serialización de un objeto Aglet:* cuando un aglet se despacha, clona o desactiva, se convierte en un flujo de bits, para poder recomponerse posteriormente. Los aglets usan ObjectSerialization de Java para transportar el estado de los agentes, intentando transportar todos los objetos que sean visibles desde el objeto aglet. Todos los objetos que van a ser serializados deben implementar java.io.Serializable o java.io.Externalizable. Si se tiene un objeto no serializable al que referencie directa o indirectamente un aglet, debe declararse la referencia como *transient*. De otro modo las operaciones anteriores fallarán y lanzarán la excepción java.io.NotSerializableException que es una subclase de java.io.IOException.class.

- *Serialización de objetos que no sean proxy:* un objeto que no sea un objeto proxy que sea visible desde un aglet, migra haciendo uso de copy. Esto quiere decir que una vez serializado, un objeto compartido por varios agentes se copia y no volverá a compartirse una vez que se han realizado las operaciones mencionadas anteriormente.

- *Serialización de objetos proxy:* cuando un objeto proxy migra, mantiene el identificador del aglet y su dirección, y restablece la referencia al objeto aglet original. El objeto AgletProxy puede mantener la referencia al aglet actual, incluso

si el proxy se transfiere a una máquina remota o se desactiva, siempre y cuando el aglet resida en la misma ubicación que el AgletProxy.

- *Uso de RMI con Objetos Remotos:* el interfaz remoto se define en el RMI y se usa para identificar los objetos remotos que poseen métodos que pueden invocarse de forma remota. El programa de cliente RMI funciona con la librería de Aglets sin necesidad de tener que hacer ninguna modificación. Es decir, cuando se despacha un aglet junto con un objeto remoto, éste se rechaza automáticamente cuando se recupera el aglet.

- *Objetos RemoteServer con RMI:* los objetos RemoteServer definidos en el RMI son objetos estacionarios, por ello no pueden transferirse con el aglet. A esto se le dará soporte en versiones futuras.

- *Objeto proxy de un aglet que ha sido despachado:* actualmente, el objeto AgletProxy no puede guardar constancia de los aglets en movimiento. Una vez que un aglet se despacha, el proxy que antes referenciaba al aglet ya no es válido.

8.5.2.2 Mensajes con aglets

Una aplicación o un aglet pueden comunicarse con otro objeto aglet mediante el paso de mensajes. Un aglet que desee hablar con otro aglet tiene que crear en primer lugar un objeto **message** y enviarselo al aglet con el que desea comunicarse. Un objeto **message** tiene como argumentos un atributo **kind** qué indica de que tipo es el mensaje que estamos enviando y un objeto arbitrario. El aglet receptor determinará que debe hacer en respuesta al mensaje chequeando en el método **Aglet.handleMessage()** el campo **kind** del mensaje recibido.

Los agles soportan los siguientes tipos de paso de mensajes:

- *Tipo inmendiato:* AgletProxy.sendMessage(Message msg).
  Un mensaje de tipo inmediato es síncrono y bloquea el receptor hasta que haya finalizado con el tratamiento del mensaje.
- *Tipo futuro:* AgletProxy.sendAsyncMessage(Message msg)
  Un mensaje de tipo futuro es asíncrono y no bloquea la ejecución actual. Este método devuelve un objeto **FutureReply**, que puede usarse para obtener la respuesta o para esperar por ella.
- *Tipo único sentido:* Aglet.sendOnewayMessage(Message msg)
  Un mensaje de tipo único sentido es asíncrono y no bloquea la ejecución actual. Se diferencia del mensaje de tipo futuro en que es colocado al final de la cola de mensajes incluso si el mensaje se envía al propio aglet. **N**o devuelve ningún valor.

8.5.2.3 Paso de mensajes de forma remota con aglets

Los aglets dan soporte al paso de mensajes de forma remota, permitiendo que los objetos aglet se puedan comunicar tanto a través de mensajes locales como remotos. Los parámetros y los objetos devueltos por los mensajes pueden ser de cualquier tipo Java

que implemente **java.io.Serializable**, realizándose la serialización y la deserialización mediante **ObjetSerialization**.

Enviar un mensaje remoto se diferencia de enviar un aglet, en que con un mensaje remoto no se realiza ninguna transferencia de código binario. El paso de mensajes de forma remota suele usarse para comunicar aglets que se encuentran en máquinas diferentes, también pude reducir el tráfico en la red, el coste de definir clases, solventando además algunos problemas de seguridad. Por otro lado al despachar un aglet se puede sacar partido de la localidad, por ejemplo de operaciones de fuera de línea o de una interacción intensa entre máquinas diferentes

8.5.3 La arquitectura del sistema

La arquitectura de los aglets está constituida por dos API y dos capas de implementación. Véase la figura 8.5.3.1.

- Aglet API
- Capa de Ejecución de Aglets – La implementación del Aglet API
- Interfaz de Transporte y Comunicación de Agentes
- Capa de Transporte

La capa de ejecución de Aglets es la implementación del Aglet API, esta API provee de la funcionalidad fundamental de los aglets permitiéndoles a estos que se creen, se traten o se envíen a una máquina remota. Esta capa define el comportamiento de *Aglet* y *AgletContext*.

La capa de transporte es la responsable de transportar un agente en la forma de un flujo de bytes que contiene las definiciones de las clases así como el estado serializado del agente. Esta capa se encuentra también definida como un API, denominado Interfaz de Transporte y de Comunicación de Agentes (ATCI de la lengua inglesa *Agent Transfer and Communication Interface*), este interfaz permite a la Capa de Ejecución de Aglets usar la capa de transporte en un modo que es totalmente independiente del protocolo.

La implementación del **ATCI** es la encargada de enviar y recibir un agente, así como de establecer una comunicación entre agentes. La implementación de Aglets actual usa el Protocolo de Transferencia de Agentes (ATP de la lengua inglesa *Agent transfer Protocol*), que es un protocolo del nivel de aplicación para la transferencia de agentes móviles. El protocolo **ATP** se encuentra modelado sobre el protocolo **HTTP** y puede usarse para transferir el contenido de un agente de una forma independiente al sistema de agentes. Además para posibilitar la comunicación entre agentes, **ATP** también da soporte al paso de mensajes.

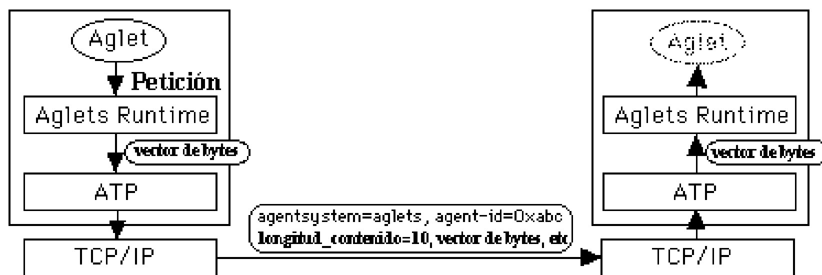


Figura 8.5.3.1 - Arquitectura de los Aglets

Cuando un aglet hace una petición de enviarse a otro destino, la solicitud se transmite a la capa de ejecución de aglets, en la que se convertirá al aglet en un vector de bytes que contenga la información del estado y del código del aglet. Si la solicitud se procesa con éxito, el aglet finalizará, seguidamente se pasará el vector de bytes a la capa **ATP** haciendo uso del **ATCI**[23], a continuación se construirá un flujo de bits que contenga información de carácter general como por ejemplo el nombre del sistema de agentes o el identificador del agente, así como el vector de bytes procedente de la Capa de Ejecución de Aglets.

8.5.3.1 El protocolo de transferencia de agentes

El protocolo de tranferencia de agentes ATP[24] (de la lengua inglesa *Agent Transfer Protocol*) es un protocolo a nivel de aplicación diseñado para transmitir un agente de una forma que sea independiente al sistema de agentes. El formato de una petición está formado por una línea de petición, campos de cabecera y un contenido. La línea de petición especifica el método de la petición, mientras que los campos de cabecera contienen los parámetros de la petición. A través de los métodos siguientes se pueden realizar cuatro peticiones diferentes como se puede ver en la siguiente figura:

Figura 8.5.3.2 - El protocolo de transferencia ATP

- *Dispatch:* el método dispatch solicita al sistema de agentes destino que reconstruya un agente a partir del contenido de la peticíón y que reinicie la ejecución del agente. Si la petición ha tenido éxito, el emisor deberá finalizar la ejecución del agente y devolver todos los recursos utilizados por éste.

---

[23] El API de la capa de transporte, ATCI, podría reemplazarse por el interfaz estándar de la MAF (Mobile Agent Facility) que está siendo propuesta actualmente al Grupo de Gestión de Objetos (OMG).

[24] El equipo de desarrollo de los aglets se encuentra actualmente trabajando en la estandarización de la capa de transferencia de agentes como una extensión del estándar **MAF** propuesto por **OMG**. Si se acepta, es posible que se cambie la interfaz **ATCI** de la capa de transporte por el de **MAF** ofreciendo una nueva capa de transporte constituida por **CORBA/IOOP** y por **ATP.**

- *Retract:* el método retract solicita a un sistema de agentes destino que devuelva un agente especificado al emisor. El receptor será el responsable de reconstruir el agente así como de reanudar su ejecución. Si el aglet se transmite con éxito, el receptor pondrá fin a la ejecución del agente y devolverá los recursos consumidos por éste.

- *Fetch:* es similar al método GET en HTTP, este método solicita al receptor que recopile y envíe cualquier información que se le especifique. (normalmente archivos .class)

- *Message:* este método se usa para enviar un mensaje a un agente especificado por el identificador del agente, devolviendo un valor como respuesta. Aunque el protocolo adopta una forma de solicitud/respuesta, no establece ningún tipo de reglas para un esquema de comunicación entre agentes.

8.5.3.2 El uso de hilos en atp

La implementación **ATP** no requiere de una creación de un nuevo hilo para manejar una solicitud de una nueva conexión. En lugar de esto el **ATP** dispone de una reserva de hilos de la que saca un hilo disponible, y lo asigna para gestionar la solicitud. Por tanto, es posible manejar varias solicitudes de forma eficiente. Puesto que un aglet posee sus propios hilos de control, se garantiza que todos los hilos usados pertenecerán al grupo de hilos del aglet y que no coinciden con los hilos en la reserva. Obsérvese la figura siguiente:

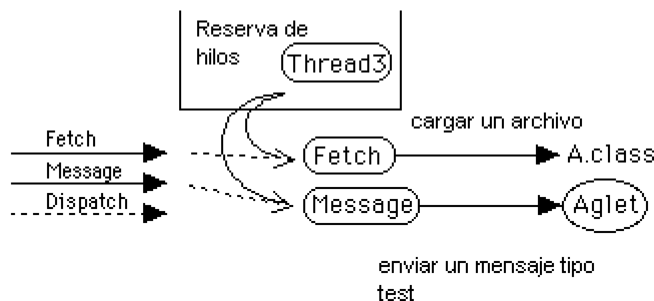


Figura 8.5.3.3 - Asignación de hilos en el protocolo ATP

8.5.3.3 Carga de clases

Con el uso de aglets, las clases que crean un aglet se determinan de forma dinámica en tiempo de ejecución. Es por esto por lo que las definiciones de clase (bytecode) del aglet tienen que cargarse en tiempo de ejecución necesariamente. Esta carga dinámica de las clases tiene lugar durante la ejecución del aglet en la máquina destino y también cuando el aglet se crea. Además en los sistemas de agentes móviles, las definiciones de clase se transmiten junto con el estado del aglet cuando este se mueve de un lugar a otro.

En Java, **el cargador de clases** que primero carga una clase se usa para cargar todas las interfaces y clases usados directamente por la clase. Al trabajar con aglets , el AgletClassLoader se usa por defecto para cargar las clases aglet. El cargador de clases

se seleccionará de acuerdo con el código base que se le pasa como parámetro al método AgletContext.createAglet(URL codigobase, String nombre, Object init) cuando se crea el aglet Es decir una nueva instancia de AgletClassLoader se crea para cada código base, los objetos aglet que tengan el mismo código base, comparten el mismo cargador de clase por regla general. El código base es un atributo inmutable de un aglet y nunca cambia durante su ciclo de vida. A continuación se mostrarán algunos ejemplos de definiciones de códigos base:

Si se le pasa como argumento el valor nulo a la primitiva AgletContext.createAglet (URL codigobase, String nombre, Object init), se buscará la clase especificada en los directorios contenidos en la variable de entorno AGLET_PATH, localizando el directorio en el que se encuentra dicha clase. No obstante si la clase se encuentra en el CLASSPATH local, el AgletClassLoader delega la solicitud al cargador de clases del sistema, usándose dicho cargador de clases para cargar la clase. Todas las clases usadas de forma directa por la clase también se cargarán haciendo uso del cargador de clases del sistema. Ha de tenerse en cuenta que esto ocurre incluso cuando la propia clase se carga. En consecuencia el aglet no tendrá un cargador de clase propio si la definición de clase del aglet se encuentra en el CLASSPATH y se cargará por tanto usando el cargador de clases del sistema.

Cuando se despacha o desactiva un aglet, el código base del aglet se almacena en un flujo y se usará para crear un nuevo cargador de clases para el aglet. Si ya existe un cargador de clases con el mismo código base en la cache, se usará éste en lugar de crear uno nuevo, para definir las clases que se reciban. Todas las clases usadas directamente por las clases definidas por el cargador de clases se cargarán o bien usando el mismo cargador de clase o el cargador de clases del sistema.

El AgletClassLoader dispone de una cache de clases en la que almacena todas las clases que se solicitan que se carguen a partir de su código base. Si la clase solicita un aglet que ya se encuentre en la cache, se usará dicha clase en lugar de bajarse una nueva clase a partir del código base, de esta forma se reduce la carga en la red y se mejora el rendimiento.

Sin embargo pueden darse casos en los que el bytecode de un aglet se haya actualizado y la nueva definición de la clase entre en conflicto con una clase almacenada en la cache del cargador de clases. Cuando un aglet se envía, el *runtime* del Aglet que recibe un nuevo aglet comprueba que las definiciones de las clases recibidas son completamente compatibles con las ya almacenadas en la cache, creándose un nuevo cargador si existe al menos una clase que entre en conflicto.

8.5.3.4 Movilidad de Clases

En los sistemas de agentes móviles, las definiciones de clase se transmiten junto con el estado del agente al mismo tiempo que el agente se mueve. Los aglets clasifican las clases de Java en tres categorías diferentes que según su código base se tratarán de forma diferente.

- *Clases del sistema:* Son las clases que se cargan a partir del CLASSPATH y por tanto no tendrán un código base.

- *Clases con código base:* Es una clase que se carga a partir del código base del aglet.
- *Otras clases:* Un aglet podrá hacer referencia a otras clases que se cargan por otros aglets desde diferentes códigos base. Esto sucede cuando un aglet recibe un mensaje que tiene como argumento un objeto perteneciente a una clase que se cargó en el código base del emisor.

Cuando un aglet se envía o se desactiva, el bytecode de clases código base que usa el aglet se almacena en forma de flujo durante el proceso ObjectSerialization, a continuación se enviará a su destino o se guardará en memoria secundaría. Por otro lado, las clases del sistema no se almacenarán en el flujo. En consecuencia nunca se enviarán al destino. Es decir, los aglets asumirá que todas las clases del sistema que use el aglet estarán también disponibles en el destino. Si alguna de estas clases no se encuentra disponible en el destino, la operación de *dispatch* fallará y se lanzará una excepción.

Además, si el usuario intenta serializar un objeto que tenga un cargador de clase distinto del cargador de clase del aglet, se generará una excepción del tipo *SecurityException*. Esto se debe a que la librería de Aglets no permite por motivos de seguridad que se cargue un aglet a partir de dos códigos base diferentes.

Cuando un aglet se activa o llega a una máquina, se recupera el bytecode del flujo, utilizándose para definir las clases necesarias para reconstruir el aglet. Si una clase local tiene el mismo nombre que otra clase en el flujo recibido, la clase local se selecciona y se usa para reconstruir el aglet. No obstante, si la clase local no es compatible con la clase original en términos de serialización, se lanzará una excepción.

Cuando se envía código binario, el *runtime* consulta al **SecurityManager** para determinar sí al aglet se le permite enviarlo. Por otra parte cuando se recibe código binario, se comprueba si está permitido enviar el código binario al runtime, para posteriormente definir la clase.

También debe tenerse en cuenta que una solicitud enviada desde el servidor para buscar un archivo de clase debe referenciarse en el SecurityManager. Si no se permite traer el fichero especificado, el SecurityManager rechazará la solicitud.

8.5.3.5 Http tunneling a través de servidores proxy

La capa ATP normalmente intenta efectuar una conexión directa entre las máquinas de la red. En casi todas las redes intranet, hay un *firewall*(cortafuegos) que evita que los usuarios abran una conexión mediante un socket directamente a un nodo externo. Esto implica que un aglet no se puede enviar o solicitar que regrese a través de un *firewall*.

Para hacerlo posible, el protocolo ATP da soporte a una técnica denominada *HTTP tunneling*. Esta técnica permite que una solicitud ATP se envíe fuera del *firewall* como una solicitud *HTTP POST* y la respuesta se recuperará como una respuesta HTTP.

Además, el servidor de Aglets tiene que ser capaz de recibir una solicitud ATP, haciendo uso de un envoltorio HTTP. Por tanto, el servidor de aglets se deberá poder configurar para recibir un mensaje HTTP POST con el tipo de contenido “x-atp” y

además debe poder desenvolver la solicitud ATP. Si se recibe una solicitud ATP en un envoltorio HTTP, la respuesta HTTP se enviará de la misma forma.

8.5.4 Seguridad en los Aglets

Los aglets tienen diferentes niveles de confianza. En la versión 1.0 del *Aglets software development Kit*, los aglets se clasifican como confiables (trusted) o no confiables (untrusted).
Se considerarán como confiables los aglets que se lancen localmente y que tengan un código base local es decir que sea nulo o un sistema de archivos local (por ejemplo: file:/Algunaruta/Algunaclase).

El resto de las aglets se tratarán como no confiables. Un ejemplo de aglets no confiables son los aglets que provienen de un servidor externo o que se generan como instancias de una clase aglet ubicada en un sitio remoto. Por tanto un aglet con un código base atp:/maquinalocal/algunaruta/algunaClase se tratará como no confiable.

Un aglet confiable puede crear otro aglet confiable en el mismo servidor aglet. En general, la configuración de acceso se especifica de forma más estricta para un aglet no confiable que para uno no confiable.

El modelo actual de IBM aglets solo diferencia entre estos dos tipos, por lo que el modelo es considerado muy deficiente, ya que no puede asignar privilegios a una autoridad en particular y menos aún a un agente específico.

8.5.4.1 La base de Datos de Políticas de seguridad

Un usuario puede especificar autorizaciones concretas para aglets confiable y no confiables. Dentro de estas especificaciones se dispone de las siguientes opciones: acceso de lectura/escritura a los archivos y librerías, instanciación de objetos, advertencias por uso de ventanas. Estas opciones de seguridad deberán definirse para cada categoría de seguridad de los aglets (*trusted* y *untrusted*). Todas estas opciones de seguridad se almacenan en los archivos:

- $HOME/.atp/security/trusted (o $JAVA_HOME si no hay $HOME)
- $HOME/.atp/security/untrusted (o $JAVA_HOME si no hay $HOME).

Estos archivos también deben tener los permisos de seguridad adecuados. De otro modo, cualquiera podría modificar los privilegios de los aglets no confialbes para que incluyesen acceso de escritura para archivos en cualquier subdirectorio. Por regla general, los aglets no deben tener acceso de escritura en los directorios: $HOME/.atp, $CLASSPATH, $AGLET_PATH y $AGLET_EXPORT_PATH.

Dentro de cada categoría (confiables y no confiables) de seguridad actualmente se da soporte a los siguientes recursos: FileSystem (Sistema de Archivos), Network (Red), Property (Propiedades) y Others (otros).

Para añadir o borrar derechos se deberá hacer únicamente usando las utilidades proporcionadas por el *Aglets software development Kit*, como lo es el Tahiti.

8.5.4.2 La clase AgletSecurityManager

El administrador de seguridad de la librería de aglets es parte del concepto de seguridad del lenguaje Java. El Administrador de Seguridad de Java contiene un cierto número de métodos que tienen como función el ser llamados para comprobar ciertas acciones que están relacionadas con la seguridad del sistema.
Por defecto, los aglets no pueden realizar la mayoría de las opciones que se acaban de describir y las ventanas que sean abiertas por aglets no confiables mostrarán una advertencia.

La clase de Java *SecurityManager* no fue pensada para utilizarse directamente. En el AWB la clase *AgletSecurityManager* hereda de la clase *SecurityManager* e implementa las políticas de seguridad deseadas. La clase *AgletSecurityManager* ofrece las siguientes comprobaciones:

Método	Descripción
checkCreateAgletContext()	Comprueba si en el hilo de ejecución actual se puede crear un nuevo agletContext
checkAgletContextAccess(AgletContext cxt)	Comprueba si el hilo de ejecución actual puede acceder a el agletContext
checkAgletAccess(Aglet aglet)	Comprueba si el aglet especificado puede accederse
checkCreate(URL agletURL, String nombre)	Comprueba si al hilo actual se le permite acceder al aglet
checkClone(Aglet aglet)	Comprueba si el hilo actual puede clonar al aglet
checkRevert(Identity quien,Aglet aglet, URL url)	Comprueba si a un aglet puede solicitársele que regrese desde una máquina remota
checkRetract(URL agletURL,AgletID aid)	Comprueba si el propio aglet puede solicitar el regresar.
checkReceive(Identity quien, Identity creador, AgletID id, URL desde)	Comprueba si el aglet puede recibirse.
checkDispose(Aglet aglet)	Comprueba si el hilo actual de ejecución puede desechar el aglet.
CheckDispatch(Aglet aglet,URL url)	Comprueba si el hilo actual de ejecución puede despachar al aglet.
CheckDeactivate(Aglet aglet, long t)	Comprueba si el hilo actual de ejecución puede desactivar al aglet
CheckActivate(AgletIdentifier agletidentifier)	Comprueba si el hilo actual de ejecución puede activar al aglet
CheckSubscribeMessage(String msg)	Comprueba si el hilo actual de ejecución puede recibir mensajes multicast
CheckMulticastMessage(Message message)	Comprueba si el hilo actual de ejecución

	puede enviar un mensaje multicast al aglet
CheckMessage(Aglet aglet,Message msg)	Comprueba si el hilo actual de ejecución puede enviar un mensaje al aglet
CheckPackageTransfer(String string)	Comprueba si el hilo de ejecución actual puede transferir un objeto con un nombre de clase determinado

8.5.5 Construcción de aplicaciones con aglets

La librería de aglets ofrece a los programadores un conjunto de APIs para construir y configurar aplicaciones usando la tecnología de aglets. Haciendo uso de estas APIs, una aplicación puede disponer de su propio visor configurado, importar una parte de las utilidades del servidor, o crear y lanzar un aglet que no tenga capacidad de servidor.

Existen varios escenarios para crear una aplicación usando la tecnología de Aglets. Mientras que un servidor puede hospedar a aglets que se estén ejecutando, un cliente puede crear y controlar un aglet de forma remota sin necesidad de ningún contexto de aglet. Por otro lado los applets pueden encontrarse en el servidor o en el cliente, aunque es imposible enviar un aglet a un applet.

8.5.5.1 Componentes con solo aglets

En el escenario típico no es necesario el uso de ningún API adicional. La aplicación estará formada por un conjunto de objetos aglet construidos sobre el Aglet API. Dicha configuración normalmente incluye un agente estacionario que dispone de acceso privilegiado a los recursos locales tales como las bases de datos y los sistemas de archivos. También se encargará de ofrecer servicio a los aglets que reciba.

Las clases utilizadas para definir el agente estacionario pueden encontrarse o bien en un disco local (CLASSPATH) o en un código base dado. Si el aglet se carga a partir del CLASSPATH, el administrador de seguridad no aplicará ninguna limitación de seguridad, el aglet tendrá todos los privilegios, es decir podrá hacer lo que le plazca. No obstante si se carga a partir de cierto código base, se le aplicarán las restricciones de seguridad apropiadas de acuerdo con su identidad.

Con esta configuración, un aglet entrante normalmente obtiene acceso a los servicios mediante el paso de mensajes, para seguidamente proseguir hacia el próximo servidor junto con los resultados obtenidos en el servidor en que se encontraba, o enviar los resultados al servidor de partida mediante paso de mensajes remotos para morir a posteriormente. El paso de mensajes también se encuentra bajo la supervisión del administrador de seguridad, por tanto un aglet receptor podrá denegar una solicitud.

El servidor también puede configurarse para que permita a un conjunto limitado de aglets (trusted o untrusted en la versión actual) acceder a los recursos y servicios locales. En este caso, por ejemplo, un aglet puede abrir una conexión directa a una base de datos y realizar una consulta **SQL**. Esto le da a los aglets una gran flexibilidad en el uso de los servicios del servidor, pero a su vez también existe el riesgo de que el aglet

pueda usar de forma errónea o abusar de estos recursos, pudiendo llegar a provocar que el sistema se comporte de forma inesperada o incluso que se llegue a caer el sistema.

8.5.5.2 Aplicaciones que incrustan un servidor de Aglets

Este escenario es una forma típica de añadir la funcionalidad de aceptar y hospedar aglets en una aplicación tradicional. Por ejemplo una simple aplicación *groupware* puede usar Aglets para ofrecer servicios y comunicarse con los miembros del grupo. No sería adecuado que estas aplicaciones usen el servidor de aglets tal y como es, incluso si el programador implementa su propio visor.

El *API Aglets Server* permite a los programadores de aglets incluir el servidor de aglets y el código necesario para arrancarlo en sus aplicaciones, así como la posibilidad de configurar en el servidor el administrador de seguridad, la persistencia, el visor, etc.
Además, es posible que una aplicación no desee que reciba a aglets mientras está creando o enviando un aglet . El Server API permite a una aplicación ejecutar la librería Aglets *runtime* sin las capacidades de demonio para este propósito.

8.5.5.3 Aplicaciones cliente

Una aplicación cliente no dispone de servicios de servidor, sólo dispone de las operaciones de comunicación para crear y controlar un aglet de forma remota, o para enviar un mensaje remoto. Por tanto, esta configuración requiere menos recursos y además reduce las amenazas de seguridad, puesto que no se baja ningún código binario, aunque todavía puede seguir sacándole provecho a las ventajas del servidor.

Por ejemplo la consola de un sistema de administración de red puede que no le sea necesario instalar la parte del servidor. La aplicación de consola, que normalmente tiene una capacidad de cliente, puede crear un aglet monitor en una máquina y dejar que un aglet “detective” viaje por varias máquinas, enviando información de vuelta a la consola.

8.5.5.4 Herramientas adicionales del AgletsFramework

El Aglets Workbench de IBM ofrece el primer entorno visual[25] para la construcción aplicaciones de red que usen agentes móviles [LANG96]. El Aglets Workbench incluirá los siguientes paquetes:

- El Aglets FrameWork para agentes móviles basados en Java.
- El Protocolo de Transmisión de Agentes (ATP).
- JDBC para DB2.
- Tahiti un administrador de agentes visual.
- Fiji un lanzador de agentes Web.

El Aglets Workbench ofrece un potente paradigma que unifica:

[25] Las herramientas visuales están en construcción y son muy elementales.

- Objetos móviles y estacionarios.
- Paso de objetos, paso de mensajes y paso de datos.
- Proceso asíncrono y síncrono
- Objetos locales y remotos.
- Operaciones en línea y fuera de línea.
- Ejecución en paralelo y secuencial.

8.5.6 Evaluación del sistema de los aglets

El sistema de agentes de los aglets puede ser empleado para desarrollar cualquier tipo de aplicación que requiera del uso de agentes móviles en redes locales o en Internet. Quizá debemos de considerar que la mayor parte de las ventajas de este sistema sobre los otros sistemas de agentes en el mundo son gracias a los beneficios que obtiene de su plataforma, ya que Java les proporciona: portabilidad, seguridad y ubicuidad.

8.5.6.1 Ventajas

- Los aglets es una interfaz de desarrollo bien diseñada. La existencia de lugares y dominios proporciona la posibilidad de modularizar un sistema complejo así como de añadir políticas de seguridad propias para cada lugar.
- Toma ventaja de las características de Java para proporcionar portabilidad, seguridad, autonomía y sobre todo ubicuidad, lo que le permite una amplia aceptación.
- Al ser basado en Java puede utilizar todas la herramientas gráficas y de bases de datos que ofrece el lenguaje, logrando con esto que el agente pueda crear interfaces de interacción gráficas así como aplicaciones que accedan a bases de datos mediante el JDBC.
- El diseño del sistema está pensado para que los agentes puedan navegar por Internet, y proporciona la posibilidad de enlazar a los aglets con navegadores.
- Proporciona primitivas simples y flexibles de migración y comunicación . De igual forma tiene una gran cantidad de servicios (que ofrece la librería) para que los agentes puedan implementar una gran cantidad de tareas mas fácilmente.
- El modelo de seguridad es bueno y cuando este terminado será un sistema completamente confiable.

8.5.6.2 Desventajas

- No conserva el estado de ejecución de agente, debido a que la máquina virtual de Java no soporta accesos a su pila. Este problema no será mejorado mientras no se modifique la máquina virtual de Java[26].
- El modelo de seguridad actual es muy genérico, ya que sólo permite agrupar a los agentes en dos categorías: confiables y no confiables, por lo que se carece de un control de seguridad más específico.

[26] Todos los sistemas de agentes móviles basados en Java tendrán el mismo problema.

- Se exige al programador muchos conocimientos para aplicaciones demasiado simples, es decir para aplicaciones demasiado simples el programador tiene que tener conocimientos del lenguaje Java así como de programación orientada a objetos.
- Es menos maduro que los otros sistemas de agentes móviles que tienen más tiempo en el mercado.

8.6 UNA ALTERNATIVA PARA LA COMUNICACIÓN ENTRE AGENTES: KQML

El lenguaje de manipulación de consultas de conocimiento [FINI95] (KQML de la lengua inglesa *Knowledge Query and Manipulation Lenguaje*) es una alternativa para gestionar muchas de las dificultades de comunicación entre agentes inteligentes, para lograrlo es necesario proporcionarles un lenguaje común. Esto significa que ellos deben compartir una sintaxis y semántica común.

Debido a que no hay un lenguaje universalmente aceptado, podemos afirmar que todos agentes pueden comunicarse uno con el otro sí ellos tienen un lenguaje de representación común o usan lenguajes que sean transportables. Si asumimos el uso de un lenguaje transportable, es necesaria para la comunicación de agentes el compartir un marco de aplicación de conocimiento con el fin de interpretar los mensajes que ellos intercambian.

KQML es un lenguaje y un conjunto de protocolos que les permite a los agentes de software la identificación, conexión con y el intercambio de información con otros agentes.

8.6.1 Una descripción de KQML

El lenguaje KQML consiste de trece capas: La capa de contenido, la capa de mensajes y la capa de comunicación. La capa de contenido soporta el contenido actual del mensaje, en el propio lenguaje de representación de los programas. KQML puede transportar expresiones codificadas en cualquier lenguaje de representación, incluyendo lenguajes expresados como cadenas ASCII y aquellos expresados mediante el uso de notaciones binarias.

La capa de comunicación codifica un conjunto de características del mensaje las cuales describen los parámetros de comunicación de más bajo nivel, como la identidad del transmisor y la de receptor, y un único identificador asociado con la comunicación.

La capa de mensajes es usada para codificar un mensaje que una aplicación le gustaría transmitir a otra. Esta capa representa el corazón del lenguaje KQML, y determina los tipos de interacciones que uno puede tener con un agente parlante de KQML. La función primaria de la capa de mensajes es identificar el protocolo que será usado para

entregar el mensaje y proporcionar una acción de comunicación o premisa que el transmisor adjunte al contenido. Esta capa también incluye características opcionales que describen el contenido de lenguaje, tales como un descriptor de nombres. Esta característica hace posible que las implementaciones de KQML analicen, enruten y entreguen los mensajes de forma apropiada aunque su contenido sea inaccesible.

La sintaxis de KQML esta basada en una lista de paréntesis balanceada. El elemento inicial es la acción a ejecutar (performative); los elementos restantes son los argumentos de la acción como los son la llave y los pares de valores. Debido a que lenguaje es relativamente simple, la sintaxis actual no es significativa y puede ser cambiada en un futuro si es necesario. La sintaxis revela los orígenes de las implementaciones iniciales, las cuales fueron hechas en *lisp*; y han resultado muy flexibles.

Un mensaje KQML de un agente *Joe* representando una consulta acerca del precio sobre la existencia de un producto de IBM podría ser codificado como :

```
(ask-one
        :sender joe
        :content (PRICE IBM ?price)
        :receiver stock-server
        :reply-with obm-stock
        :language LPROLOG
        :ontology NYSE_TICKS)
```

En este mensaje, la acción ejecutada es *ask-one*, el contenido es (*price ibm ?price*) la antología asumida por la consulta es identificada por la partícula *nyse-ticks*, el receptor del mensaje en su servidor identificado como *stock-server* y la consulta esta escritura en un lenguaje llamado LPROLOG.

Existe una gran variedad de procesos internos en los protocolos de intercambio información en KQML. En el más simple, un agente actúa como un cliente y envía una consulta a otro agente que actúa como un servidor, luego espera por una respuesta, como es mostrado en la figura 8.6.1. La respuesta del servidor podría consistir de una respuesta simple, de una colección o de un conjunto de respuestas asíncronas. En otro caso común, mostrado entre los agentes A y C, la respuesta del servidor no es una respuesta completa pero sí un gestor que permite al cliente preguntar por los componentes de la respuesta de uno por uno. Un ejemplo común de este intercambio ocurre cuando un cliente consulta una base datos relacional la cual produce una secuencia de instancias como respuesta. En este caso las transacciones individuales implican una comunicación síncrona entre los agentes.

Un caso diferente ocurre cuando el cliente se suscribe a la salida de un servidor y un número indefinido respuestas asíncronas le llegan en intervalos irregulares de tiempo, como es el caso de los agentes A y D. El cliente no sabe cuándo llegará cada mensaje de respuesta y puede estar ocupado ejecutando algunas otras tareas cuando la respuestas lleguen.

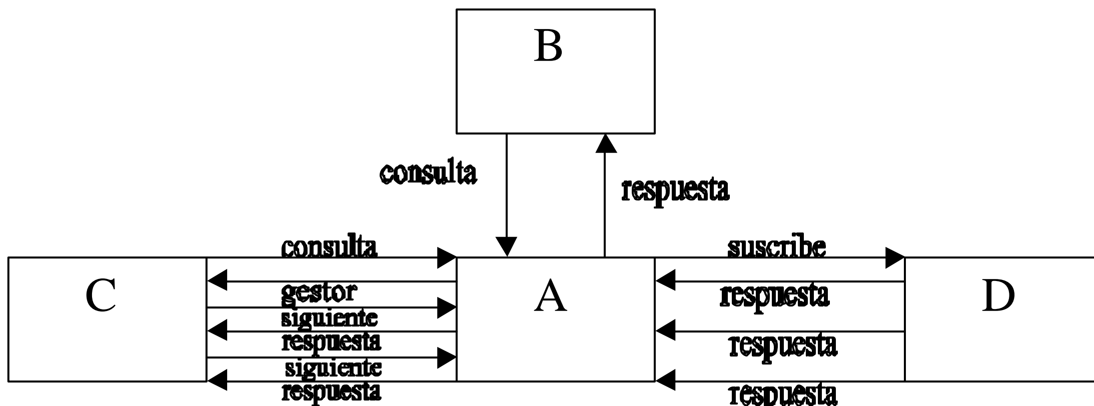


Figura 8.6.1 - Diversos protocolos de comunicación básicos soportados por KQML

8.5.2 Facilitadores y el entorno de agentes parlantes de KQML

Un facilitador (facilitator) es un agente que ejecuta varios servicios útiles de comunicación, por ejemplo el mantenimiento de un registro de servicio de nombres, el envío de mensajes a los servicios de nombres, enrutar mensajes basados en su contenido proporcionando la coincidencia por servicios (matchmaking) entre información proporcionada por los proveedores y la información requerida por los clientes, y proveyendo la mediación y la interpretación de servicios.

Una de las principales funciones de los agentes facilitadores es ayudar a otros agentes a encontrar los clientes y servidores apropiados. El problema de cómo los agentes encuentran a un facilitador en principio no concierne directamente a KQML y tiene una variedad diversas soluciones.

Las aplicaciones actuales de usado una de dos simples técnicas. La primera solución es que todo los agentes usan un facilitador común central cuya localización es un parámetro de iniciación cuando los agentes son lanzados. La otra alternativa es que encontrar y establecer contacto con un facilitador local era una de las funciones de la API de KQML.

Por convención un agente facilitador debería estar corriendo en una máquina servidora con una dirección simbólica *facilatator.domain* y escuchar al puerto estándar de KQML.

8.6.3 Una breve evaluación de KQML como lenguaje de comunicación de agentes.

Una evaluación básica de lenguaje como se encuentra hoy en día consiste en analizar algunas características básicas:

- *Forma:* Los mensajes de KQML son flujos lineales de caracteres con una sintaxis como la de Lisp. La sintaxis es simple y permite la adición de nuevos parámetros si es necesario en futuras revisiones de lenguaje.

- *Contenido:* las implementaciones actuales del lenguaje no soportan contenidos que no sean ASCII, y no hay nada en el lenguaje que prevenga el soporte. El lenguaje ofrece un conjunto mínimo de acciones a ejecutar que cubren un repertorio básico de acciones de comunicación, y se pretende mantener un punto medio en el número de acciones que se incluyan .
- *Semántica:* la semántica es aún un tema abierto y aún no ha sido bien definido.
- *Implementación:* las dos implementaciones actuales de las interfaces de KQML, la *LockHeed KQML API* y la *UBMC KQML API* proveen un enrutado de mensajes independiente del contenido y un facilitador. La eficiencia de la comunicación ha sido investigada y varias mejoras de compresión han sido añadidas las cuales reducen los costos de comunicación al minimizar el tamaño de los mensajes y también al minimizar una fracción sustancial del símbolo de búsqueda.
- *Trabajo en red:* los agentes parlantes de KQML pueden comunicarse directamente con otros agentes dentro de una red a través de su dirección o por medio de un nombre simbólico.
- *Entorno:* puede usar cualquier protocolo transporte así como cualquier mecanismo de transporte. No hay más restricciones en la capa de contenido de lenguaje que la provisión de funciones que gestionen las acciones a ejecutar por la misma capa de contenido de lenguaje de la aplicación.
- *Seguridad:* los temas de seguridad y autenticación no han sido implementados apropiadamente por la comunidad de KQML. No se ha hecho una decisión sobre si ellos debieran gestionarlos en el nivel del protocolo de transporte o en el nivel de lenguaje. Si se hace al nivel de lenguaje, nuevas acciones a ejecutar o parámetros del mensaje pueden ser introducidos para que se permita la encriptación del contenido o de todo el mensaje de KQML.

8.6.4 Aplicaciones de KQML

El lenguaje KQML y las implementaciones del protocolo han sido usadas en varios prototipos y sistemas de demostración. Las aplicaciones han abarcado desde diseño concurrente e ingeniería de sistemas de hardware y software, hasta la planeación y programación de actividades de lógica de transportación militar, arquitecturas flexibles para sistemas de información heterogéneos de gran escala, integración de software basado agentes y la planeación y recuperación de información en sistemas cooperativos.

KQML tiene el potencial para mejorar significativamente las capacidades y la funcionalidad de la integración en gran escala, así como los esfuerzos de interoperabilidad que se encuentra hoy en día bajo las tecnologías de comunicación e información tales como la infraestructura de información nacional y CORBA de OMG, de igual forma en áreas de aplicación de comercio electrónico, sistemas de información de salud e integración de empresas virtuales.

8.7 FACILIDAD DE INTEROPERABILIDAD DE SISTEMAS DE AGENTES MÓVILES DE CORBA (MASIF): UN INTENTO DE ESTANDARIZACIÓN

Una meta importante de la tecnología de agentes móviles es la interoperabilidad entre sistemas de agentes de diversos fabricantes. Cuando el sistema de agentes origen y el sistema destino son similares, la estandarización de estas acciones puede resultar en interoperabilidad, si son completamente diferentes no podrá ser lograda.

La MASIF (también es llamada MAF un acrónimo de la propuesta original *Mobile Agent Facility)* es un estándar de OMG para sistemas de agentes móviles.

8.7.1 Interoperabilidad

La MASIF trata de la interoperabilidad entre los sistemas de agentes escritos en el mismo lenguaje pero potencialmente por un vendedor diferente. La MASIF no estandariza las operaciones locales de agentes como interpretación, serialización, ejecución y deserialización [OMG97]. Estas acciones son implementadas específicamente, y no hay razones para limitar las implementaciones a una arquitectura individual.

El objetivo de la MASIF es acelerar el uso de los sistemas de agentes móviles al maximizar la interoperabilidad entre los sistemas de agentes móviles mientras se minimiza el impacto de los elementos estandarizados en un sistema particular.

8.7.1.1 ¿Qué se debe estandarizar ahora?

Existen varias áreas de la tecnología de agentes móviles que la comunidad de agentes móviles debería estandarizar ahora para promover la interoperabilidad:

- *Gestión del agente:* debería ser posible crear un agente, suspender su ejecución, reanudar su hilo, y terminarlo en una forma estándar.
- *Transparencia del agente:* permitir que un agente origen viaje a un sistema de agentes destino permite tomar beneficios de la localidad y solo se logrará si la transferencia es de forma estándar.
- *Nombres de agente y del sistema de agentes:* cuando se invoca una operación de gestión, el agente y el sistema de agentes gestionados deben ser identificados. Por tanto la sintaxis de los nombres se debe estandarizar para que se logre dicha identificación.
- *El tipo de los sistemas agentes y sintaxis de la localización*: la sintaxis de la localización debe ser estandarizada para que un agente pueda acceder a la información que define el tipo del sistema de agente, desde el sistema de agentes destino deseado. Es importante que una autoridad de nombrado sea definida para cada sistema agentes.

8.7.1.2 ¿Qué se debe estandarizar más tarde ?

La comunidad de agentes móviles pasará un tiempo estandarizando la seguridad de los agentes móviles hasta que la seguridad de los sistemas de superar el problema

Cuando los formatos de serialización para el código de un agente y estado de ejecución sean similares; será posible crear agentes estándar entre diferentes tipos de sistemas de agentes. En conclusión podemos decir que la comunidad de agentes deberá estandarizar las interfaces de seguridad y de serialización posteriormente.

8.7.1.3 Sumario de la interoperabilidad de la MASIF.

La MASIF debe llevar a cabo la interoperabilidad definiendo interfaces para acciones tales como la suspensión, reanudación, y terminación de agentes. Esta interoperabilidad está relativamente dirigida a que se implemente por la mayoría de los sistemas de agentes.

El historial (*tracking*) de un agente permite seguir la trayectoria de los agentes registrados con *MAFFinders*(por ejemplo servicios de nombrado) de diferentes sistemas de agentes. La comunicación entre los agentes esta fuera del alcance de la esta especificación .

La MASIF lleva a cabo el transporte de agentes definiendo métodos par la recepción de agentes y la búsqueda de sus clases. La interoperabilidad del transporte de agentes no es tan simple de lograr y requiere cierta cantidad de cooperación entre los diseñadores de diversos sistemas de agentes.

8.7.2 Conceptos básicos

Se definen algunos de los conceptos principales en la terminología de agentes móviles que posteriormente serán usados para describir las interfaces. Se usa terminología de programación orientada a objetos para definir conceptos de agentes móviles.

8.7.2.1 Autoridad del agente

La autoridad del agente identifica a la persona u organización para la que el agente actúa. Una autoridad debe ser autentificada.

8.7.2.2 Nombres del agente

la identidad de un agente es un valor único dentro del alcance de la autoridad que identifica a una instancia particular de un agente. La combinación de la autoridad del agente, identidad y tipo del sistema de agente son siempre un valor único globalmente.

8.7.2.3 Localización del agente

La localización del agente es la dirección del lugar. Un lugar reside dentro de un sistema de agentes.

8.7.2.4 Sistema de agentes

Un sistema de agentes es una plataforma que puede crear, interpretar, ejecutar, transferir y terminar agentes. Como un agente, el sistema de agentes esta asociado con una autoridad. Un sistema de agentes es identificado únicamente por sus nombre y su dirección. Un servidor puede contener uno o más sistemas de agentes.

8.7.2.5 Tipo del sistema de agentes

El tipo de un sistema de agentes describe el perfil de un agente. Por lo tanto, un cliente que requiere una función de un sistema de agentes debe especificar el perfil del agente (tipo del sistema de agentes, lenguaje, método de serialización) para identificar de forma única la funcionalidad deseada.

8.7.2.6 Interconexión de sistema de agentes a sistemas de agentes

Toda la comunicación entre sistemas de agentes es a través de la infraestructura de comunicación (CI). La región de administración define servicios de comunicación para las comunicaciones dentro y fuera de la región.

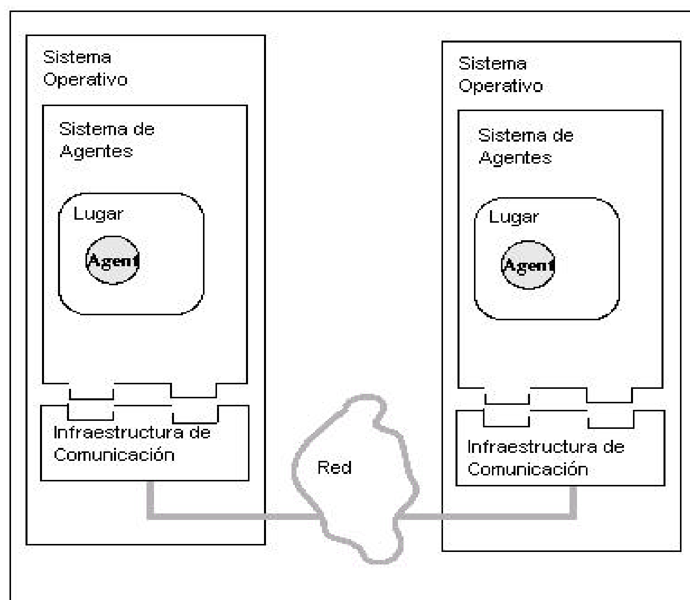


Figura 8.7.2.1 - Interconexión entre sistemas de agentes

8.7.2.7 Lugar

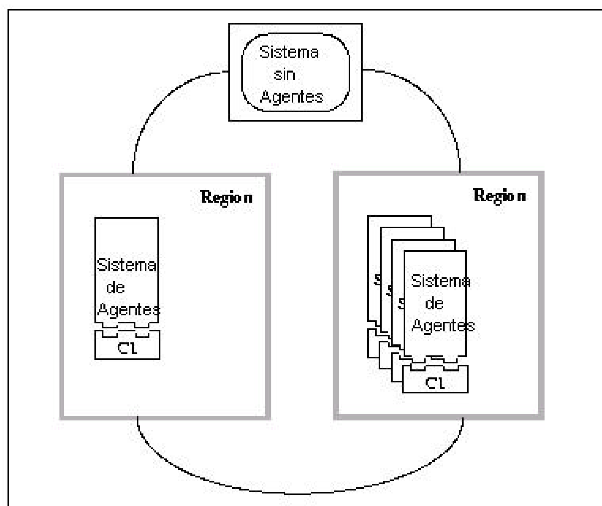
Un lugar es un contexto dentro de un sistema de agentes en el cual un agente se puede ejecutar. Este contexto puede proveer funciones tales como el control de acceso. Un sistema de agentes puede contener uno o más lugares y cada lugar puede contener uno o más agentes. Cuando un cliente requiere la localización de un agente, este recibe la dirección del lugar donde el agente se está ejecutando .

8.7.2.8 Regiones

Una región es un conjunto de sistemas de agentes que tienen la misma autoridad, pero no necesariamente el mismo tipo que de sistema agentes. Una región provee un nivel de abstracción a los clientes para que se comuniquen desde otras regiones. Cada región contiene uno o más puntos de acceso a la región y por estos medios, las regiones están interconectadas para formar una red.

8.7.2.9 interconexión de región a región

Las regiones son interconectadas a través de una o más redes y pueden compartir un servicio de nombres basado en un acuerdo entre las autoridades de cada región y las implementaciones específicas de estas regiones. Un sistema que no sea de agentes también puede comunicarse con los sistemas de agentes dentro de cualquier región siempre y cuando el sistema que no es de agentes tenga la autorización para hacerlo.



8.7.2.2 - Interconexión entre regiones

8.7.2.10 Serialización de serialización

La serialización es el caso de almacenar al agente en una forma serializada. La deserialización es el proceso de restaurar al agente de su forma serializada .

8.7.2.11 Código base

El código base especifica la localización de las clases usadas por el agente. El código puede ser sistemas de agentes o objetos que no sean CORBA tales como servidores de Web.

8.7.2.12 Infraestructura de comunicación

Una infraestructuras de comunicación que provee servicios de transporte de comunicaciones, nombrado, y servicios de seguridad para un sistema de agentes.

8.7.2.13 Localidad

En el contexto de agentes móviles, la localidad es definida como las cercanía al sistema de agentes destino, en el mismo servidor o en la misma red.

8.7.3 Interacción de Agentes

Existen tres tipos de interacciones de agentes que están relacionadas con la interoperabilidad.

8.7.3.1 Creación de agentes remoto

En la creación de agentes remota, un cliente interacciona con el sistema de agentes destino para requerirle que una clase particular de agente sea creado. El cliente se autentifica a sí mismo con el sistema de agentes destino, y establece la autoridad y las credenciales de seguridad que el nuevo agente poseerá. El nuevo agente tendrá generalmente la misma autoridad que el sistema de agentes.

8.7.3.2 Transferencia de agentes

Cuando un agente desea transferirse a otro sistema de agentes, el sistema de agentes crea una solicitud de viaje. Como parte de la solicitud de respuesta, el agente ofrece información del nombre y la dirección que sirve para identificar el lugar de destino. Cuando un sistema de agentes destino accede a una transferencia, el estado del agente, la autoridad, las credenciales de seguridad y en caso de que sea necesario su código se transferirá al sistema de agentes destino. El sistema de agentes entonces reactiva el agente y reanuda su ejecución.

8.7.3.3 Invocación de métodos del agente

Un agente puede invocar un método de otro agente u objeto, si está autorizado y tiene una referencia al objeto. Cuando un método es invocado, la información de seguridad proveída a la infraestructura de comunicaciones que ejecuta la invocación del método debe ser la autoridad del agente.

8.7.4 Funciones de un sistema de agentes

Entre las acciones comunes de los sistemas de agentes se encuentran: la transferencia del agente, la recepción del agente, la creación, y la provisión de un identificador único globalmente, etc.

8.7.4.1 Transferencia de un agente

Lo primero que un agente debe hacer es identificar su destino, después debe solicitar al sistema de agentes origen su transferencia al sistema de agentes destino. Cuando el sistema de agentes destino recibe la petición de traslado del agente, las siguientes acciones son iniciadas:

- Suspender al agente.
- Identificar las clases del estado del agente que serán transferidas.

- Serializar la instancia de la clase del agente y el estado
- Codificar el agente serializado para el protocolo de transporte escogido.
- Autentificar al cliente.
- Transferir al agente.

Antes de que el agente sea recibido por el sistema de agentes destino, el sistema de agentes destino debe determinar si puede interpretar al agente. Si el sistema de agentes puede interpretar al agente, lo acepta y entonces:

- Autentificar al cliente.
- Decodificar al agente.
- Deserializar la clase y el estado del agente.
- Hacer una instancia del agente.
- Restaurar el estado del agente.
- Restaurar el estado de ejecución.

Una vez que el agente es transferido es necesario transferir sus clases de un sistema de agentes a otro. Hay tres razones del porque es necesario transferir las clases de un agente móvil durante el termino de su vida:

- *La creación de una instancia como parte de la creación de un agente remoto:* la clase del agente es requerida para su creación.
- *La creación de una instancia como parte de la transferencia del agente:* la clase del agente es requerida para la creación de su instancia en el sitio remoto.
- *Ejecución del agente después de la creación de su instancia:* la creación de otros objetos, requiere obviamente las clases de estos objetos.

El modelo conceptual común es lo suficientemente flexible para soportar variaciones de la transferencia de clases para que los fabricantes tengan más de un método disponible. El modelo soporta:

- Transferencia automática de todas las clases posibles.
- Transferencia automática de sólo las clases del agente, las otras clases son transferidas en demanda.
- Transferencia automática de las clases del agente, transferencia en demanda de todas las otras clases cuando se transfiere un agente, unido con todas la transferencia de todas la clases automáticamente cuando se crea al agente remotamente.
- Transferir una lista de los nombres de todas las clases posibles en la creación del agente o en la solicitud de transferencia.

8.7.4.2 Creación de un agente

Para crear un agente, un sistema de agentes debería:

- Comenzar un hilo del agente.
- Crear una instancia de la clase del agente.
- Generar (si es necesario) y asignar un nombre de agente único globalmente que pueda ser autentificado.

- Comenzar la ejecución del agente dentro de su hilo.

8.7.4.3 Provisión de nombres y localizaciones únicas globalmente

Un sistema de agentes debe generar un nombre único para sí mismo, y para los lugares que él crea. Tendrá también que generar un nombre único para los agentes que él crea si es necesario.

8.7.4.4 Soportar el concepto de una región

Un sistema de agentes soporta una región al cooperar con otros sistemas de agentes de la misma autoridad y al soportar un punto de acceso a la región.

8.7.4.5 Encontrar un agente móvil

Cuando un agente quiere comunicarse con otro agente, debe ser posible encontrar el sistema de agentes destino para establecer la comunicación. El sistema de agentes puede proveer un servicio de nombrado basado en el nombre de los agentes.

8.7.4.6 Asegurar un entorno seguro para las operaciones de los agentes

Para asegurar la seguridad de los recursos del sistema, un sistema de agentes debe identificar y verificar la autoridad que envía al agente. El sistema de agentes debe también saber que tipo de acceso le es permitido a la autoridad. Otro aspecto de importante de la seguridad es la confidencialidad.

Tanto los agentes como los sistemas de agentes deben tener definidas políticas de seguridad. Las políticas de seguridad contienen reglas para varios propósitos, incluyendo:

- Restricción o aceptación de las capacidades del agente.
- Establecer los limites para la consumo de recursos del agente.
- Restringir o aceptar su acceso.

Cuando un agente invoca una operación o desea viajar, el agente especifica sus requerimientos para la calidad de la seguridad de la comunicación de la red. Estos requerimientos incluyen:

- Confidencialidad.
- Integridad.
- Autenticación.
- Detección de replicas.

8.7.4.7 Autenticación

Para satisfacer la políticas de seguridad del sistema de agentes destino, la mutua autenticación de los sistemas de agentes es requerida. Los sistemas de agentes operan sin supervisión humana. Por lo tanto, los sistemas de agentes que participan en la transferencia del agente deben participar en el proceso de autenticación sin intervención humana.

La autenticación del agente difiere de la del sistema de agentes. La autenticación del agente usa verificadores (authentificators). Un verificador es un algoritmo que determina la autenticidad del agente. Un verificador usa información tal como la autenticidad del sistema de agentes origen o del cliente lanzado, la autoridad del agente y del sistema de agentes involucrado, y posiblemente información acerca de cuales autoridades son confiables con el fin de autentificar al agente.

El administrador de la región puede imponer limitaciones a los recursos en agentes o código que se origine de fuentes no confiables.

El nivel de confianza asignado a un agente está parcialmente en función de la autoridad del agente y si la autoridad fue autenticada. El nivel de confianza debe también depender de firmas digitales, o de otras técnicas.

Probablemente los requerimientos de servicios de seguridad sean el mayor desafío en la tecnología de agentes. Los requerimientos para las comunicaciones seguras de agentes móviles son:

- *Autenticación del cliente para la creación de agentes remotos*: este proceso puede hacerse mediante contraseñas o tarjetas inteligentes.
- *Autenticación mutua de sistemas de agentes*: normalmente se logra probando que el sistema de agentes se encuentra en posesión de alguna información secreta como por ejemplo un sistema de clave privada.
- *Acceso de los sistemas de agentes para la autenticación de resultados y credenciales:* cuando tiene lugar la comunicación entre agentes, el sistema de agentes destino puede ser capaz de comprobar las credenciales del agente y del sistema de agentes fuente para así poder verificar su autenticidad. El resultado de la autenticación determina que política de seguridad aplicar.
- *Autenticación y delegación de agentes*: si un agente migra a otro sistema de agentes, las credenciales deben transferirse con el agente si la migración tiene éxito. Si el sistema de agentes hace una llamada RPC en nombre del agente cliente, el servidor debe ser capaz de pasar las credenciales de seguridad del agente cliente.
- *Políticas de seguridad de los agentes y de los sistemas de agentes*: un agente debe ser capaz de controlar el acceso a su métodos. El agente y su sistema de agentes asociado debe ser capaces de establecer y reforzar sus controles de acceso tanto a métodos como a recursos.
- *Integridad, confidencialidad, detección de respuesta y autenticación*: Para cualquier comunicación el solicitante debe ser capaz de especificar sus requerimientos de integridad, confidencialidad, detección de respuesta y autenticación.

8.7.5 IDL de la MASIF

La MASIF es una colección de definiciones e interfaces que proveen una interfaz interoperable para los sistemas de agentes móviles.

El módulo de la MASIF contiene dos interfaces, obsérvese la figura 8.7.5.1:

- La interfaz *MAFAgentSystem* que define la operaciones de gestión de los agentes, incluyendo la recepción, la creación, la suspención y la terminación de ellos. También incluye un método para obtener una referencia al *MAFFinder.*

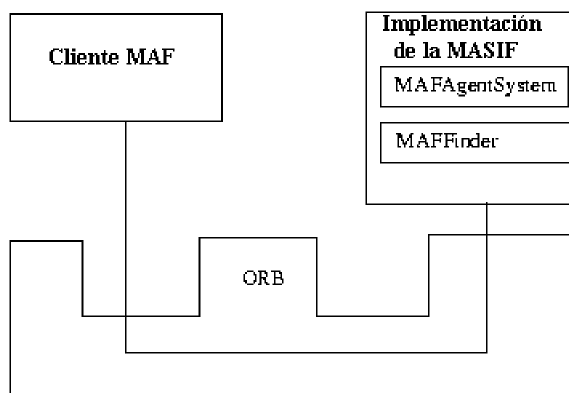


Figura 8.7.5.1 - Relación entre la MASIF y un ORB

- La interfaz *MAFFinder* que define las operaciones para el registro, para eliminar el registro, y localizar agentes, lugares y sistema de agentes. La *MAFFinder* es un servicio de nombrado. Puede ser compartida entre regiones. Sin embargo, por simplicidad de definición, hay un *MAFFinder* para cada región. Antes que un cliente pueda pedir al *MAFFinder* que encuentre un objeto, el cliente debe obtener la referencia al objeto del *MAFFinder.* Para obtener la referencia a dicho objeto, el cliente puede usar el servicio de nombrado de CORBA o el método *AgentSystem.get_MAFFinder().*

8.7.5.1 Nombre, nombres de clase y localización

Para que sea posible la localización de agentes, es necesario estandarizar algunos conceptos que serán utilizados por los métodos de las interfaces. De esta forma se hacen las siguientes definiciones:

```
typedef short AgentSystemType;
typedef sequence<octet> OctetString;
struct ClassName{
        string name;
        OctetString discriminator;
};
typedef sequence<ClassName> ClassNameList;
typedef sequence<OctetString> OctetStrings;
typedef OctetString Authority;
typedef OctetString Identity;
struct Name{
```

```
        Authority authority;
        Identity identity;
        AgentSystemType agent_system_type;
};
typedef string Location;
```

En el módulo MAF, el nombre es definido como una estructura que consiste de tres atributos: autoridad, identidad, y *agent_system_type*. Estos atributos crean un nombre globalmente único para un agente o un sistema de agentes.

Cuando nombre es un sistema de agente, el *agent_system_type* es el tipo del sistema de agentes que generó la identidad del agente. Cuando nombre es un nombre de un sistema de agentes, el *agent_system_type* es el tipo del sistema de agentes. CORBA usa el termino principal en lugar de autoridad.

La estructura de nombre define la sintaxis para un agente o para el nombre de sistema de agentes.

La estructura *ClassName* define la sintaxis para el nombre de una clase. El nombre de un clase tiene un nombre humanamente leíble y una cadena de caracteres (string) que asegura que el nombre de una clase es única dentro de su ámbito.

Esta especificación no posee ningún mecanismo para hacer que el nombre de una clase sea globalmente única. Por lo que los fabricantes serán los encargados de solucionar este problema.

Si los administradores de la región de los sistemas de agentes que se comunican acuerdan un esquema de nombrado de clases globalmente único, el problema de nombres duplicados puede ser evitado.

8.7.5.2 Localización

En la interfaz *MAFFinder*, la localización especifica la ruta de un sistema de agentes basado en el nombre de un sistema de agentes, agente o lugar. Una vez que el cliente obtiene la localización del sistema de agentes, éste debe convertir la localización (una cadena de caracteres) a la referencia del objeto del sistema de agentes para invocar las operaciones ofrecidas.

La cadena de caracteres de la localización esta en una de estos formatos:

- Un URI (Universal Resource Identifier) que contiene un nombre CORBA.
- Una URL (Universal Resource Location)que contiene una dirección Internet.

La ventaja de usar el servicio de nombrado de CORBA es que es independiente del protocolo. La ventaja de usar una dirección Internet es que ésta es más apropiada para los agentes móviles en Internet.

Para determinar en que formato esta la localización, el cliente analiza la cadena de caracteres hasta la primera columna (:). Si los caracteres que preceden la columna son

"CosNaming", la cadena de caracteres es un nombre CORBA. Si los caracteres que preceden la columna son "mafiiop", la cadena de caracteres es una dirección Internet.

8.7.5.3 Servicio de nombrado de OMG de identificadores de autoridad

Los identificadores asignados a los parámetros tales como el tipo del sistema de agentes y verificador deben ser únicos a través de todas las implementaciones de la MASIF. Por lo tanto, es necesario que alguien asigne y mantenga estos parámetros. OMG es propuesto para que sea la autoridad de nombrado para los parámetros de la MASIF que requiere unicidad global. OMG debería asignar valores y gestionar los parámetros de las siguientes definiciones:

```
typedef short LanguageID;
typedef short AgentSystemType;
typedef short Authenticator;
typedef short SerializationID;
typedef sequence<SerializationID> SerializationIDList;
```

These parameters are used in the following definitions:

```
typedef any Property;
typedef sequence<Property> PropertyList;
struct LanguageMap {
	LanguageID language_id;
	SerializationIDList serializations;
};
typedef sequence<LanguageMap> LanguageMapList;
struct AgentSystemInfo {
	Name system_name;
	AgentSystemType system_type;
	LanguageMapList language_maps;
	string system_description;
	short major_version;
	short minor_version;
	PropertyList properties;
};
struct AuthInfo { //authentication information
	boolean is_auth;
	Authenticator authenticator;
};
struct AgentProfile {
	LanguageID language_id;
	AgentSystemType agent_system_type;
	string agent_system_description;
	short major_version;
	short minor_version;
	SerializationID serialization;
	PropertyList properties;
};
```

Esta especificación no señala que tipos de sistemas de agentes, lenguajes, mecanismos de serialización y métodos de autenticación deben ser usados para incorporar nuevos sistemas.

8.7.6 La interfaz MAFAgentSystem

La interfaz *MAFAgentSystem* define métodos de objetos que soportan tareas de gestión de agentes tales como buscar un nombre de un sistema de agentes y recibir un agente. Estos métodos y objetos proveen un conjunto básico de operaciones para la transferencia del agente. Los métodos y las interfaces son las siguientes:

```
interface MAFAgentSystem {
    Name create_agent(
        in Name agent_name,
        in AgentProfile agent_profile,
        in OctetString agent,
        in string place_name,
        in Arguments arguments,
        in ClassNameList class_names,
        in string code_base,
        in MAFAgentSystem class_provider) raises (ClassUnknown,
        ArgumentInvalid, DeserializationFailed, MAFExtendedException);
    OctetStrings fetch_class(in ClassNameList class_name_list, in string
      code_base, in AgentProfile agent_profile) raises (ClassUnknown,
      MAFExtendedException);
    Location find_nearby_agent_system_of_profile (in AgentProfile profile)
      raises (EntryNotFound);
    AgentStatus get_agent_status(in Name agent_name) raises
      (AgentNotFound);
    AgentSystemInfo get_agent_system_info();
    AuthInfo get_authinfo(in Name agent_name) raises (AgentNotFound);
    MAFFinder get_MAFFinder() raises (FinderNotFound);
    NameList list_all_agents();
    NameList list_all_agents_of_authority(in Authority authority);
    Locations list_all_places();
    void receive_agent(
        in Name agent_name,
        in AgentProfile agent_profile,
        in OctetString agent,
        in string place_name,
        in ClassNameList class_names,
        in string code_base,
        in MAFAgentSystem agent_sender) raises (
        ClassUnknown, DeserializationFailed, MAFExtendedException);
    void resume_agent(in Name agent_name) raises (AgentNotFound,
      ResumeFailed, AgentIsRunning);
    void suspend_agent(in Name agent_name) raises (AgentNotFound,
      SuspendFailed, AgentIsSuspended);
    void terminate_agent(in Name agent_name) raises (AgentNotFound,
      TerminateFailed);
    void terminate_agent_system() raises (TerminationFailed);
};
```

A continuación describiremos la función de los todos los métodos de esta interfaz.

8.7.6.1 create_agent()

Un sistema de agentes ejecuta la operación *create_agent()* para crear un agente de acuerdo a la petición remota del cliente. El nombre real del agente creado es regresado. Este puede ser el mismo que es dado como parámetro si el cliente es responsable del nombrado.

8.7.6.2 fetch class()

El método *fetch_class()* regresa las definiciones de una o más clases. En el caso de los sistemas de agentes no orientados a objetos, este método es usado para buscar código.

8.7.6.3 find_nearby_agent_system_of_profile()

Este método pide la *MAFFinder* que encuentre un sistema de agentes cercano que pueda ejecutar el agente que el cliente quiera enviar. La responsabilidad de encontrar el sistema de agentes del tipo correcto recae en el *MAFFinder*.

8.7.6.4 get_agent_status()

El método *get_agent_status()* regresa el estado (estatus) del agente especificado.

8.7.6.5 get_agent_system_info()

Este método regresa la estructura *AgentSystemInfo*. Esta estructura contiene información a cerca del sistema de agentes, incluyendo el nombre y el perfil que soporte.

8.7.6.6 get_authinfo()

Este método retorna información a cerca de sí un agente fue autenticado, y que método de autenticación fue usado.

8.7.6.7 get_MAFFinder()

Regresa una referencia al MAFFinder para la localización de agentes, lugares, y sistemas de agentes.

8.7.6.8 list_all_agents()

Proporciona la lista de todos los agentes registrados dentro del sistema de agentes.

8.7.6.9 list_all_agents_of_authority()

Lista todos los agentes dentro del sistema de agentes que tiene la autoridad especificada.

8.7.6.10 list_all_places()

Proporciona una lista de todos los lugares dentro del sistema de agentes.

8.7.6.11 receive_agent()

Un sistema de agentes usa este método para recibir una instancia de un agente.

8.7.6.12 resume_agent()

Este método reanuda la ejecución del agente especificado.

8.7.6.13 suspend_agent()

Suspende la ejecución del agente especificado.

8.7.6.14 terminate_agent()

Detiene la ejecución de agente especificado.

8.7.6.15 terminate_agent_system()

Detiene la ejecución del sistema de agentes.

8.7.7 La interfaz MAFFinder

La interfaz *MAFFinder* provee métodos para mantener una base de datos dinámica de nombres y localizaciones, lugares y sistema de agentes. La interfaz no establece que método usa un cliente para encontrar a un agente. En vez de ello, proporciona formas

para localizar agentes, sistemas de agentes, y lugares con una amplia gama de técnicas de localización.

Aunque la interfaz *MAFFinder* no restringe a los fabricantes a un cierto conjunto de esquemas de localización de agentes, ésta asume una estructura de bases de datos que soporta registro, eliminación de registros, y localización de agentes, sistemas de agentes, y lugares. Los métodos de la interfaz son los siguientes:

```
interface MAFFinder {
      void register_agent (
          in Name agent_name,
          in Location agent_location,
          in AgentProfile agent_profile,) raises (NameInvalid);
      void register_agent_system (
          in Name agent_system_name,
          in Location agent_system_location,
          in AgentSystemInfo agent_system_info) raises (NameInvalid);
      void register_place (
          in string place_name,
          in Location place_location) raises (NameInvalid);
      Locations lookup_agent (
          in Name agent_name,
          in AgentProfile agent_profile) raises (EntryNotFound);
      Locations lookup_agent_system (
          in Name agent_system_name,
          in AgentSystemInfo agent_system_info)
          raises (EntryNotFound);
      void unregister_agent (in Name agent_name)
          raises (EntryNotFound);
      void unregister_agent_system (in Name agent_system_name)
          raises (EntryNotFound);
      void unregister_place (in string place_name)
          raises (EntryNotFound);
};
```

A continuación describiré los métodos de la interfaz

8.7.7.1 lookup_agent()

Retorna la localización de los agentes especificados. Este método puede buscar un agente especifico por nombre, o puede buscar un conjunto de agentes que coincidan con un perfil de agente especifico.

8.7.7.2 lookup_agent_system ()

Retorna la localización de un sistema de agentes registrado con el *MAFFinder*. El método puede buscar por un sistema de agentes específico por nombre, o puede buscar por un conjunto de sistemas de agentes que coincidan con los parámetros especificados en *AgentSystemInfo.*

8.7.7.3 lookup_place()

Regresa la localización de un lugar registrado en el *MAFFinder*.

8.7.7.4 register_agent ()

Añade el nombre del agente a la lista de los agentes registrados con el *MAFFinder*.

8.7.7.5 register_agent_system ()

Este método añade el nombre del sistema de agentes para la lista de sistemas registrados con el *MAFFinder*. Como un sistema de agentes es un objeto estacionario, la interfaz *MAFFinder* no permite invocaciones múltiples de esta operación con el mismo. Cuando se mueve un sistema de agentes, debe eliminarse su registro antes de registrarlo otra vez en la nueva localidad.

8.7.7.6 register_place ()

Añade el nombre del lugar nombrado a la lista de lugares registrados en el *MAFFinder*. Un lugar es un objeto estacionario y no se soportan invocaciones múltiples de este método con el mismo nombre.

8.7.7.7 unregister_agent ()

Elimina el agente especificado de la lista de agentes que están registrados en el *MAFFinder*.

8.7.7.8 unregister_agent_system ()

Elimina el sistema de agentes especificado de la lista de sistema de agentes que están registrados en el *MAFFinder*.

8.7.7.9 unregister_agent ()

Elimina el lugar especificado de la lista de lugares que están registrados en el *MAFFinder*.

8.8 COMPARACIÓN, CARENCIAS Y TENDENCIAS DE LOS SISTEMAS DE AGENTES MÓVILES

El estado del arte de esta tecnología permite vislumbrar los avances que han tenido los sistemas de agentes, las necesidades que se han superado y retos que faltan por resolver de forma satisfactoria. De igual forma se observe una tendencia hacia los cuales se dirigen todos, con el afán de abarcar el amplio mundo de Internet.

8.8.1 Comparación

En términos generales podríamos decir que D'Agents sería apropiado para aplicaciones pequeñas debido a las limitaciones del lenguaje *Tcl*[27], Ara por su parte parece ser el indicado para todas las aplicaciones que requieran ser programadas en más de un lenguaje[28] y con un desempeño aceptable. Finalmente los IBM aglets resultan ser los mejores para grandes aplicaciones que se deseen usar en Internet, ya que tienen toda la funcionalidad que el lenguaje Java puede ofrecer.

Es difícil realizar una comparación de los sistema de agentes presentados, ya que difieren en su arquitectura y plataformas de desarrollo, no obstante presentaré una tabla comparativa de las características que son fundamentales en un sistema de agentes para poder visualizar los servicios que cada uno nos puede proporcionar, así como su grado de eficacia.

Es importante hacer mención que todas las características descritas son con las versiones públicas actuales de los productos hasta la fecha de publicación de este trabajo. Ya que algunos sistemas han sido modificados y están por sacar nuevas versiones que ofrecen mas características.

Sistema de Agentes	Portabilidad	Seguridad	Trans del estado de ejecución	Lenguajes	Mensajes Remotos	Interfaz con el usuario	Velocidad
D'Agents	Unix	Buena	Si	Tcl, (Java, python)[29]	Si	Muy buena	Regular
Ara	Unix	Buena	Si	Tcl, C	No	Buena	Buena
Aglets	Todas	Regular	No	Java	Si	Buena	Muy buena

Aunque cada plataforma tiene ventajas y desventajas, considero que las características de Java y su ubicuidad le permite a los IBM aglets ofrecer una mayor funcionalidad, aunado a esto los fines comerciales de la plataforma han hecho que exista todo un equipo de trabajo en IBM Japón que están trabajando en la creación de un entorno completo para el desarrollo de agentes móviles, que en un futuro se ofrecerá al mercado.

Por su parte OMG ha creado una infraestructura bien diseñada para la interoperabilidad de agentes móviles que soporta asignación de nombres, seguridad y localización que son indispensables para que los agentes naveguen en Internet [PERE98a], sin embargo al no estandarizar la interpretación y serialización de los agentes, la interoperabilidad en ocasiones se logrará parcialmente. Mas aún la MASIF no da ningún soporte para la comunicación mediante mensajes lo que podría ser una alternativa para la interoperabilidad de sistemas de agentes que no fueran del mismo tipo.

[27] El leguaje no es orientado a objetos

[28] Se supone que la actual implementación de *Agent Tcl: D'Agents* soporta varios lenguajes, sin embargo no se ha hecho pública. Una vez que sea pública también podría ser apropiada para estas aplicaciones.

[29] Ya están disponibles en sus versiones internas

8.8.2 Carencias

Podemos considerar que los sistemas de agentes han superado en gran medida la primera necesidad que debe satisfacer la tecnología "**portabilidad**", la llegada de Java ha contribuido en gran medida a solucionar dicho problema, sin embargo quedan muchos requerimientos por satisfacer para que la tecnología pueda proveer un entorno completo de desarrollo de aplicaciones. Para conocer todos los requerimientos que un sistema de agentes debe tener consulte el capítulo X. Por ahora para concluir este tema mencionaré a continuación algunas de las carencias más importantes que los sistemas de agentes suelen tener.

8.8.2.1 Seguridad

La seguridad, factor fundamental para que la tecnología de agentes móviles sea aceptada en Internet no ha sido cubierta satisfactoriamente. D'Agents no proporciona un mecanismo en el que se puedan asignar privilegios de forma específica a un agente. Ara tiene un mecanismo de autenticación poco eficiente y los Aglets solo son capaces de restringir el acceso de los agentes generalizando en dos grupos en confiables y en no confiables.

Los sistemas de agentes móviles deberán introducir nuevos métodos de seguridad que garanticen una completa integridad de la información. De igual forma estos métodos deberán poder personalizar para que cada usuario pueda configurarlo de una forma que satisfaga sus necesidades.

Actualmente los sistemas carecen de métodos de autenticación con firmas digitales, tampoco utilizan la criptografía para transferir la información de forma encriptada. Finalmente carecen de bases de datos en donde se puedan dar privilegios específicos a un agente para acceder a los recursos del sistema.

8.8.2.2 Interoperabilidad

La MASIF logra una interoperabilidad parcial de los sistemas de agentes, es decir si los sistemas de agentes son del mismo tipo se puede lograr una buena interoperabilidad, pero si por el contrario son de tipo diferente la interoperabilidad es casi nula.

A pesar de la existencia de la MASIF, ésta solo aplica para el mundo CORBA, por lo que sí agentes móviles del sistema de los Aglets quisieran interactuar con agentes móviles del sistema MOLE[30] no podrían hacerlo, el único medio de interacción sería la implementación de las interfaces que especifica la MASIF de CORBA, lo cual implica una ardua labor.

La aparición de Java produjo la aparición de una gran cantidad de sistemas de agentes móviles basados en Java, con la rapidez que evoluciona esta tecnología, en poco tiempo

[30] Es otro sistema de agentes móviles basado en Java

el mundo de Internet estará lleno de subredes cada una con un tipo diferente de sistema de agentes (probablemente basado en Java) pero con poca capacidad de interacción entre ellos, lo que menguará el poder de las aplicaciones desarrolladas.

8.8.2.3 Entornos visuales de desarrollo e interfaces gráficas del usuario

Actualmente los sistemas de agentes móviles no cuentan con un entorno visual de desarrollo, lo que exige a los desarrolladores tener un nivel de conocimientos (de programación) informáticos avanzado.

Si observamos la mayor parte de las aplicaciones de los agentes móviles son enfocadas a la búsqueda y recopilados de datos en redes, dichas tareas son requeridas por todo tipo de usuarios y por tanto cualquier tipo de usuario debe ser capaz de crearlos. Por tanto se requiere un tipo de desarrollador de aplicaciones que todo usuario pueda utilizar.

De igual forma los agentes deben ser capaces de utilizar interfaces gráficas que les permita interactuar con los usuarios de una forma más sencilla, para que estos puedan seleccionar el tipo de tarea e información que requieran.

8.8.2.4 Incorporación a los navegadores

Uno de los objetivos principales de los agentes móviles es la búsqueda y recopilación de datos en Internet, sin embargo actualmente no se encuentran incorporados a los navegadores lo que facilitaría la creación de búsquedas asíncronas por parte de los usuarios de Internet.

8.8.3 Tendencias

La tecnología de agentes móviles tomó gran fuerza desde la aparición de Java, este lenguaje proporcionaba portabilidad y muchas características que lo hacían ideal para el desarrollo de sistemas de agentes. Desde entonces han aparecido una gran cantidad de sistemas de agentes móviles basados en Java (refiérase a la sección 6.4.2).

Los sistemas que ya existían anteriormente como D'Agents y Ara están modificando su núcleo para soportar a Java como uno más de los lenguajes en los que se puedan programar los agentes.

Los sistemas de agentes que logren una mayor interoperabilidad en la MASIF serán aquellos que estén implementados en Java por la facilidades de interacción que brinda el

lenguaje. Actualmente no ha habido ninguna implementación de un ORB[31] que siga completa y adecuadamente la especificación de la MASIF para dar soporte a la movilidad.

Desde que Java esta siendo usado en todas partes será más fácil utilizar un sistema de agentes móviles basado en Java que un ORB que implemente la especificación de la MASIF, Java tiene pues no sólo la ventaja de ubicuidad/portabilidad sino también la ventaja de tiempo, esta claro Java será la tendencia dominante para los sistemas de agentes móviles en los próximos meses [PERE98b].

[31] Que sea compatible con CORBA

Capítulo IX

INTEGRACIÓN DE LOS AGENTES MOVILES A SISTEMAS ORIENTADOS A OBJETOS

Por todo lo que se ha mencionado en los capítulos anteriores podemos apreciar que la tecnología de agentes esta evolucionando rápidamente y que cada día podrá brindar más y mejores soluciones a las necesidades que el mercado reclama.

Sin embargo como es obvio suponer esta tecnología no sólo servirá para implementar sistemas completos basados en ella, sino que se irá adentrando como una tecnología complementaría o de apoyo para los sistemas Orientados a Objetos (SOO) convencionales, para que realice las tareas especificas para las cuales fue desarrollada y en donde se puede obtener el mayor provecho de ella.

En la mayoría de los casos los sistemas de agentes han sido implementados con tecnologías de orientación a objetos por lo que no habrá ningún problema al añadirlos como un agregado al análisis y diseño de los SOO convencionales. Y Más aún para los sistemas de agentes basados en Java, para que un programa logre tener agentes móviles bastará con heredar de una clase.

Para la integración de los agentes móviles a los SOO habrá dos[32] principales alternativas como lo son los sistemas de agentes basados en Java o como lo es la facilidad común ofrecida por CORBA. Ambas alternativas en teoría pueden convergir a una sola ya que la facilidad común que se ha implementado por OMG ha sido tan bien diseñada que ofrece un enlace para cualquier tipo de sistemas de agentes, no sólo para los basados en Java. La convergencia hacia CORBA dependerá de su éxito[33] en el mercado.

[32] Existen otras alternativas como lo son KQML, TCL, etc., que son lenguajes propietarios por lo que sería más difícil la integración de agentes mediante estos lenguajes y por lo cual no los considero.

[33] Con esto me refiero al dominio que CORBA pueda llegar a tener en el mercado y ver si es capaz de acaparar la mayor parte del desarrollo de software futuro mediante sus ORBs.

Es decir los agentes móviles incorporados a SOO desarrollados en Java que utilicen un sistema de agentes móviles basado en Java podrán comunicarse con los agentes móviles incorporados a un SOO distribuido implementado mediante un ORB CORBA mediante la Facilidad común para agentes móviles MAF (del ingles Mobile Agent Facility) que ha sido desarrollada para CORBA. Esto se logrará si el sistema de agentes que desea comunicarse cumple con las especificaciones mínimas para lograr el enlace.

9.1 ¿CÓMO INTEGRAR AGENTES MÓVILES EN UN SISTEMA ORIENTADO A OBJETOS?

Cuando se desea incorporar la tecnología de agentes móviles a un Sistema Orientado a Objetos, lo más común será incorporarla como un componente agregado que forme parte del sistema total, es decir, debe usarse la agregación para que integre todos los componentes que forman parte del sistema en un todo.

Así pues imaginemos el esquema inicial de un sistema de gestión, podría ser de la siguiente manera:

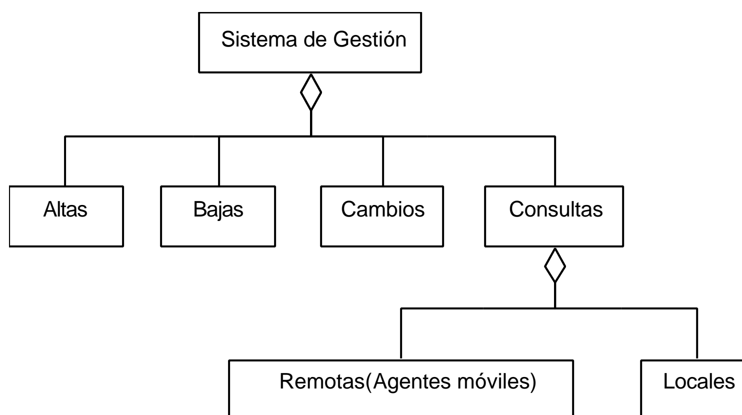


Figura 9.1.1 – Agregación de un Componente de Agentes Móviles para hacer Consultas Remotas.

Es obvio suponer que el componente de agentes será el encargado de realizar todas las consultas remotas o aquellas tareas en dónde se saque el mayor provecho de la tecnología. En este ejemplo esta claro que será mucho más rápido enviar a un agente móvil al sitio remoto para que recopile la información deseada y después regrese con los datos, de esta forma evitará las demoras producidas por el tráfico en la red.

Supongamos ahora que agregamos un componente de ventas por Internet al sistema anterior, compuesto a su vez de dos módulos, uno a través de paginas WEB y otro por medio de un mercado electrónico que sea gestionado por agentes móviles.

En este sistema (Figura 9.1.2) el módulo de mercado de agentes tendrá acceso a las bases de datos del sistema para poder gestionar la compra y venta de productos. Las interfaces necesarias para realizar estas tareas las heredaría de una clase mercado, que proveería todas estas características.

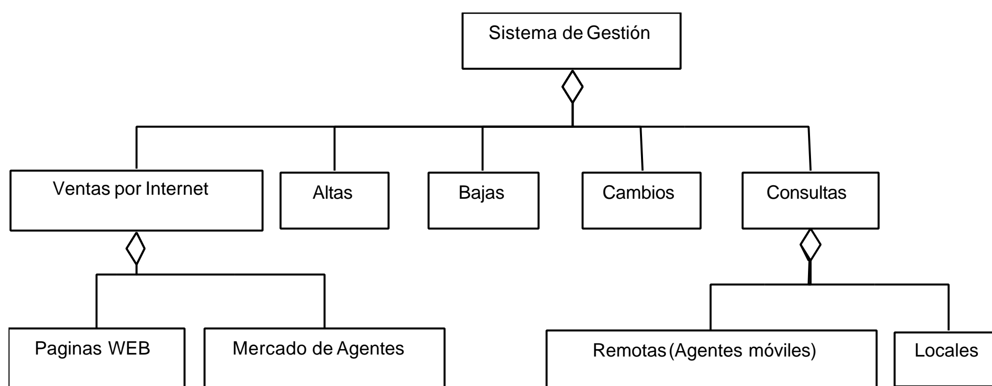


Figura 9.1.2 – Agregación de un componente de agentes móviles para vender productos en un mercado electrónico.

9.2 ¿CÓMO REPRESENTAR AGENTES MÓVILES EN UML?

Ciertamente al trabajar con sistemas de agentes móviles o bien integrarlos a sistemas orientados a objetos nos es necesario representarlos de una forma estándar y en una notación orientada a objetos estándar.

Debido a la importancia que UML ha ido adquiriendo en los últimos tiempos y que ha llegado a ser considerado una fusión/estándar en las notaciones de Sistemas Orientados a Objetos, utilizaré esta nomenclatura para definir una forma de representación para agentes móviles así como para desarrollar este trabajo.

No pretendo tratar de proponer una nueva metodología que no sería interpretada apropiadamente por otras personas sino simplemente usar las herramientas que UML ofrece que son los estereotipos.

Un estereotipo [CUEV98a] es una forma de clasificar las clases a alto nivel. Éstos se muestran con un texto entre doble ángulo << nombre >>, por ejemplo: << agente remoto >>.

Los estereotipos también se pueden definir con un icono, o bien se puede razonar que los tipos clase, asociación, y generalización son subtipos de estereotipos del metamodelo. De igual forma los estereotipos también se pueden colocar a grupos de la lista de elementos de una clase.

Al hacer uso de agentes móviles podemos reconocer varios tipos de agentes que pueden interaccionar dentro de un sistema y que por consiguiente requieren una representación específica con el fin de poder ser interpretados.

Así pues he reconocido 3 tipos de agentes móviles fundamentales que son:

- *Agentes locales:* este tipo de agentes son los que son creados dentro de ese servidor de agentes, y son considerados confiables por ser locales. La mayor parte de los agentes serán de este tipo por lo que al representarlo bastará con indicar en el

estereotipo la palabra **agente**. Opcionalmente también podría indicarse como **agente local**.

Figura 9.2.1 – Representación de Agentes Locales en UML

- *Agentes remotos:* estos agentes son los que llegan a un servidor a través de la red y que han sido creados en otros servidores. Dentro del estereotipo se especificará **agente remoto** y se añadirá después de dos puntos o bien la autoridad o la dirección IP de donde provienen. La autoridad se refiere al usuario o empresa propietaria del agente. Al ser creados en otra máquina son considerados como no confiables.

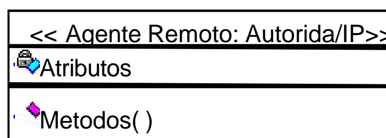


Figura 9.2.2 – Representación de Agentes Remotos en UML

- *Agentes clónicos*: son agentes que son copias idénticas de otros agentes, de hecho podría haber agentes clónicos-remotos, sin embargo desde el momento que es remoto para el nuevo servidor resulta intrascendente si es clónico o no y por consiguiente solo es considerado como remoto. Sin embargo localmente es importante poder distinguir si un agente es una copia o no de otro. Sobre todo cuando se considera que un agente padre puede crear a varios agentes hijos idénticos con el fin de realizar una tarea común en paralelo. Este tipo de agentes llevarán en el estereotipo la frase **agente clónico** y después de dos puntos el nombre del agente del cual fueron copiados.

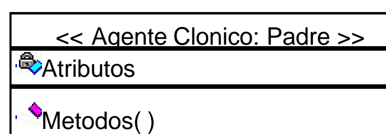


Figura 9.2.3 – Representación de Agentes Clónicos en UML

Podrían distinguirse otro tipo de agentes, como agentes inactivos[34] sin embargo al ser almacenados a disco el nombre de la base de datos puede servir de representación del agente dentro del modelo.

De igual forma los estereotipos pueden usarse para definir relaciones o bien representar la interacción entre los agentes.

[34] Aquellos que son guardados a una base de datos temporalmente

9.3 CLASIFICACIÓN DE PATRONES DE DISEÑO DE AGENTES MÓVILES

Se han considerado tres clases de patrones fundamentales [PERE98f] para el diseño de agentes[35] que son: *migración (traveling)*, *tarea* e *interacción.* Éste esquema de clasificación hace más fácil el entender el dominio y aplicación de cada patrón, el distinguir los diversos patrones y el descubrir nuevos patrones.

9.3.2 Patrones de Migración

La migración es la esencia de los agentes móviles. Los patrones de migración tratan sobre diversos aspectos de la gestión de movimientos de los agentes móviles, como lo son el enrutado(routing) y la calidad de servicio. Estos patrones nos permitirán reforzar la gestión de la movilidad.

- *Itinerario:* mantiene una lista de destinos, define un esquema de enrutado, y controla casos especiales tales como, qué sucede si un destino no existe, y siempre sabe a donde irá en el siguiente traslado. De igual forma permite el rehuso de itinerarios ya que son guardados como agendas.
- *Envío(forward):* provee una forma para que un servidor envíe automáticamente a un agente que acaba de llegar a otro servidor.
- *Acceso(ticket):* especifica una dirección destino y encapsula la calidad del servicio y permisos necesitados para enviar a un agente a la dirección del servidor y entonces ejecutarlo ahí.

A continuación presentaremos un ejemplo del patrón de *itinerario*[36] que es quizá el más importante de este grupo.

Siendo una entidad móvil y autónoma, un agente es capaz de navegar independientemente a diversos servidores. También debe ser capaz de gestionar excepciones tales al tratar de enviarse a otro sitio, como puede ser el desconocimiento de un servidor.

Por la tanto será preferible separar la gestión de la navegación del comportamiento del agente y de la gestión de los mensajes. La idea es de cambiar la responsabilidad de navegación del objeto agente a un objeto itinerario. La clase *itinerario* proveerá de una interfaz para mantener modificable el itinerario de los agentes y despacharlo a nuevos destinos.

El agente creará el objeto *itinerario* y lo inicializará con una lista de destinos que visitará secuencialmente y con una referencia al agente. Entonces, el agente usará el método *go()* para enviarse a sí mismo al siguiente destino disponible o regresar a su

[35] Esta clasificación es quizá la más importante de las que se han presentado hasta la fecha ya que agrupa perfectamente las tres funciones principales involucradas con los agentes.
[36] Este patrón ha sido implementado ya en algunos sistemas como Odyssey con su clase worker, que define un itinerario de destinos y tareas definidas.

origen. Para lograrlo es necesario que el objeto itinerario sea transferido junto con el agente. Es patrón puede apreciarse en la siguiente figura:

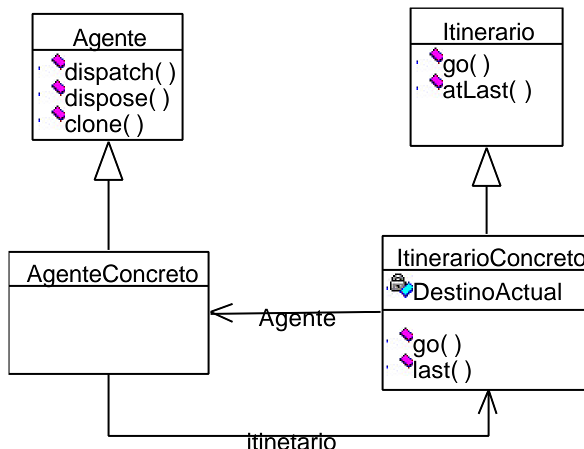


Figura 9.3.1.1 – Patrón de Itinerario

El objeto *ItinerarioConcreto* guarda el destino actual del agente concreto y puede despacharlo a nuevos destinos. Este patrón debe usarse cuando se desea:

- Proveer una interfaz uniforme para el traslado de agentes en forma secuencial a múltiples servidores.
- Para definir recorridos que pueden ser compartidos por varios agentes.

9.3.2 Patrones de Tareas

Los patrones de tareas se refieren al análisis de tareas y como estas tareas son delegadas a uno o más agentes. En general, las tareas pueden ser dinámicamente asignadas a agentes de propósito general. Inclusive, una tarea dada puede ser ejecutada por un a agente individual o por varios agentes trabajando en paralelo y cooperando para lograrla. Existen dos patrones de tareas fundamentales:

- *Maestro-Esclavo:* define un esquema con el cual un agente maestro puedo delegar una tarea a un agente esclavo. El agente esclavo se desplazada al servidor especifico, ejecutará la tarea y regresará con el posible resultado.
- *Plan:* provee una forma de definir la coordinación de múltiples tareas, para ser ejecutadas en múltiples servidores secuencialmente o en paralelo por varios agentes. El plan encapsula el flujo de tareas y promueve su rehuso así como la asignación dinámica de las mismas.

Mostraremos un ejemplo del patrón *maestro-esclavo* que es quizá el más útil del grupo de tareas.

Existen varias razones por las que un agente(maestro) crearía a otros agentes (esclavos) y les delegaría tareas. La principal es, obviamente el poder delegar tareas a otros agentes

para que estas se ejecuten en paralelo mientras el agente maestro continua con otras labores o bien gestiona la información recibida de sus agentes esclavos.

La idea de este patrón es crear clases abstractas *Maestro* y *esclavo*, para localizar las partes invariantes de la delegación de tareas. Los agentes maestro y esclavo son definidos como subclases de *Maestro* y *Esclavo*, en los cuales sólo varia la parte de la tarea que ejecutará y como el agente maestro debe gestionar el resultado de la tarea implementada como se aprecia en la figura 9.3.2.1.

En práctica la clase *Maestro* tiene un método abstracto *getresult()* que va a ser sobrescrito para definir como gestionar el resultado de la tarea. La clase *Esclavo* tiene dos métodos abstractos *initializeJob* and *doJob* que inicializa los pasos a ejecutar antes de que el agente viaje a un nuevo destino y para concretar la tarea respectivamente.

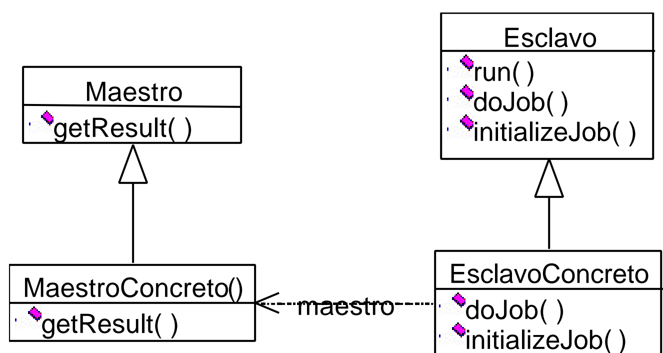


Figura 9.3.2.1 – Patrón Maestro-Esclavo

Este patrón provee una forma fundamental de reutilizar código entre clases de agentes y debe usarse principalmente en los siguientes casos:

- Cuando un agente necesita desempeñar una tarea in paralelo con otras tareas de las cuales es responsable.
- Cuando un agente estacionario quiere desempeñar una tarea en un destino remoto.

9.3.3 Patrones de interacción

Los patrones de interacción se refieren a la localización de agentes y a la facilitación de sus interacciones [ARID98]. Así pues, en este grupo podemos mencionar:

- *Reunión(Meeting):* proporcionan una forma para que dos o más agnetes inicien una interacción local en un servidor dado. A los agentes les es avisada la llagada de sus contrapartes para que inicie la interacción.
- *Locker:* define un espacio de almacenamiento especifico para que un agente deje datos antes de ser enviado temporalmente a otro destino y pueda evitar el trasladar datos que no va a utilizar en ese momento.
- *Mensajero:* define un agente subordinado para llevar un mensaje de un agente a otro, mientras el agente principal continua con su tarea.

- *Ayudante(facilitator):* Define un agente que provee servicios de nombrado y localización de agentes con capacidades específicas.
- *Grupo Organizado:* Crea grupos de agentes en los cuales todos los miembros de un grupo viajan juntos.

Analizaré un ejemplo del patrón de reunión, que es quizá un patrón básico para una de las utilidades mas de moda de los agentes móviles, como lo es el comercio electrónico[37].

Este patrón es aplicable siempre que sea requerido que los agentes provenientes de varios servidores interactúen localmente.

Considérese por ejemplo, agentes comerciales creados por distintos clientes y en diferentes destinos para buscar por, comprar y vender bienes particulares en nombre de sus clientes. Para lograrlo, los agentes compradores y vendedores deben interactuar extensivamente para tratar de llegar a un arreglo.

El problema principal es como sincronizar estos agentes comerciales, los cuales se encuentran inicialmente en servidores diferentes, de tal forma que ellos puedan visitar el comercio virtual y encontrarse uno a otro. El patrón de *Reunión* provee una solución a problemas de este tipo. Éste usa una clase *Reunión* que encapsula un destino específico (el lugar de encuentro) y un identificador único.

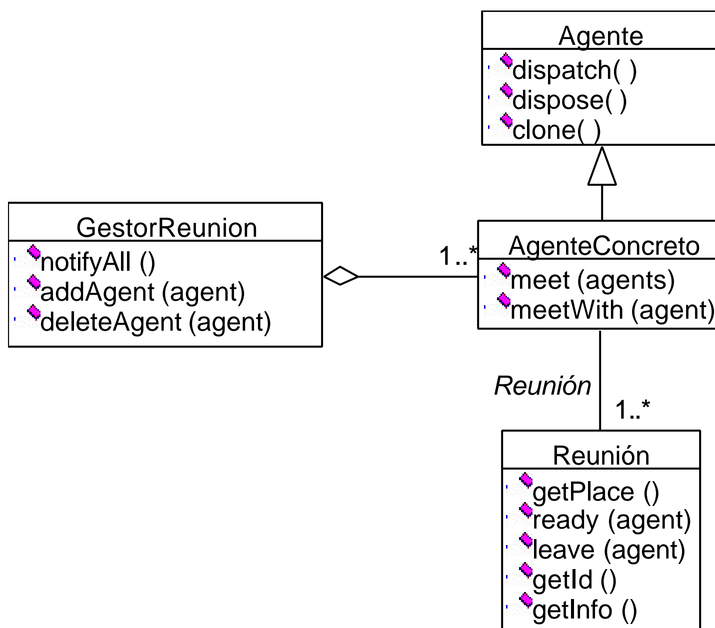


Figura 9.3.3.1 – Patrón de Reunión

Los agentes que necesiten interactuar unos con otros estarán equipados con un objeto *Reunión*. Ellos usarán el identificador único para localizar un objeto *GestorReunión* para registrarse a sí mismo. Entonces el objeto gestor de reunión notificará a los agentes

[37] Existen diversos grupos de investigación en el mundo que están desarrollando las primeras implementaciones de comercio electrónico.

que ya se encuentran registrados de la llegada del nuevo agente, éste obtendrá una referencia y viceversa para que puedan iniciarse las interacciones.

Los agentes deberán borrar su registro por ellos mismos antes de abandonar el lugar de reunión. Debido a que poseen un identificador único varias reuniones pueden ejecutarse simultáneamente en el mismo servidor. Es esquema básico de los componentes de este patrón se muestra en la figura 9.3.3.1.

Este patrón se aplica cuando es requerido que los agentes de diferentes servidores interactuen en un mismo sitio localmente; por ejemplo:

- Cuando los agentes necesitan interactuar y el costo de viajar a un lugar común es menos que el asociado con una comunicación remota.
- Cuando los agentes necesitan acceder a servicios locales en un servidor específico.

9.4 INTERACCIÓN ENTRE AGENTES Y ACCESO A BASES DE DATOS PARA SISTEMAS DE AGENTES MÓVILES BASADOS EN JAVA

En la guerra por dominar los mercados de los sistemas de agentes móviles, parece que los sistemas basados en Java le llevan la delantera a CORBA. fundamentalmente porque mientras la MAF de CORBA apenas ha sido aprobada (y aún no implementada), la plataforma Java cada día se extiende y se consolida más en el mundo

Un agente móvil creado en un sistema de agentes basado en Java en la mayoría de los casos hereda la movilidad (los métodos que le permiten migrar a otro servidor) y la capacidad para comunicarse, de una clase base que contiene la mayoría de las interfaces requeridas. Estas le permitirán enviar mensajes a otros agentes así como migrar de un servidor a otro hasta terminar un itinerario específico.

La interacción de los agentes dependerá del sistema en particular del que se hable, se puede mencionar que normalmente se realiza la comunicación entre agentes por medio de mensajes, la posibilidad de que un agente pueda crear a otro (y de eliminarlo posteriormente), así como la posibilidad de migrar a otro servidor con todo y sus datos.

De igual forma el agente podrá crear objetos, y acceder a los datos que le sean permitidos por el gestor de seguridad que se encuentra en el servidor de agentes móviles. Éste gestor será el encargado de autentificar a un agente que acabe de llegar al servidor y él será prácticamente quien decidirá los recursos[38] a los cuales el agente podrá acceder en el sistema de ficheros remoto.

Si se desea incorporar la tecnología de agentes a SOO actuales, la clave para lograrlo será el acceso a sus bases de datos actuales, esto para los sistemas de agentes basados en Java puede ser implementado a través del JDBC (Java Data Base Connectivity), que es

[38] Se refiere a la parte de los sistemas de ficheros que incluyen bases de datos, así como la posibilidad de ejecutar comandos en el servidor remoto.

la interfaz SQL para Java que ejecuta comandos SQL en los registros y recupera los resultados.

El JDBC tiene dos grupos principales de interfaces. El JDBC API que permite al programador de una aplicación abrir conexiones a una base de datos particular, ejecutar comandos SQL, y procesar los resultados. El JDBC Driver API[39] que permite al programador crear controladores basados en Java para conectarse a bases de datos SQL.

El JDBC gestiona la conexión a un servidor SQL y es requerido por todas las aplicaciones de bases de datos que involucren un servidor de datos. El JDBC es una interfaz para los programadores de aplicaciones para acceder a fuentes remotas de datos.

De esta forma un servidor de agentes móviles basado en Java, puede acceder a todos los recursos que ofrece el lenguaje, entre ellos al JDBC, de esta manera un agente móvil podrá acceder a bases de datos utilizando a su servidor de Agentes y al JDBC de Java como un puente para obtener los datos requeridos. El agente móvil [AGLE]deberá de ser capaz de usar una API para acceder a bases de datos locales sin importar el esquema y formato de almacenamiento que este debajo de ellas. Este se muestra en la siguiente figura:

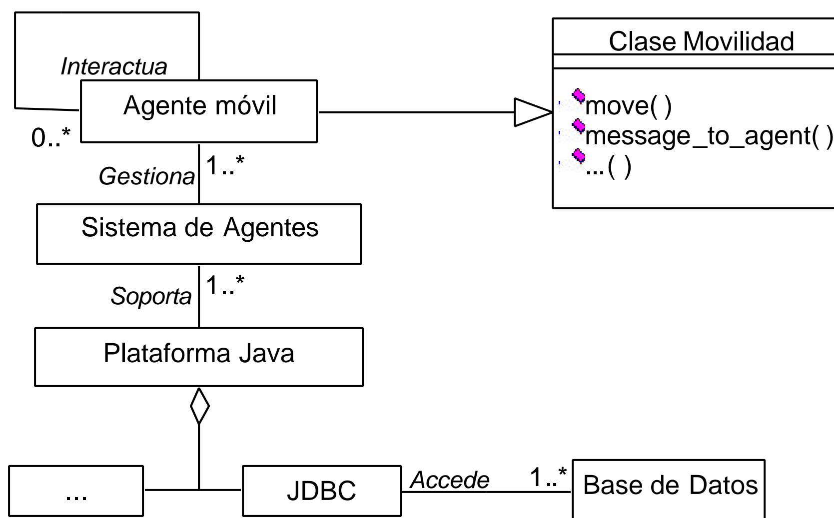


Figura 9.4.1 – Interacción de Agentes Móviles en Sistemas basados en Java

De esta forma se logra que un agente móvil pueda acceder a datos que se encuentran en bases de datos en cualquier formato[40] que sea soportado por los controladores del JDBC. Si la conexión al servidor de base de datos es a través de un controlador ODBC[41], entonces se requiere el puente de software JDBC-ODBC de JavaSoft.

[39] Ambos el JDBC API y el JDBC Driver API es expresado como un conjunto de interfaces abstractas de Java.

[40] Siempre y cuando el Sistema de seguridad (security manager) del sistema de agentes lo permita.

[41] ODBC de Object Data Base Connectivity, se utiliza para enlazar bases de datos en aplicaciones windows.

9.5 INCORPORACIÓN DE AGENTES MÓVILES A SOO CON TECNOLOGIA CORBA

Inicialmente para lograr usar agentes móviles en sistema distribuido CORBA, lo primero que se requiere es que el ORB en el cual se desarrolla el sistema tenga una implementación de la MAF.

La MAF estará formada por sus dos interfaces *MAFAgentSystem* y *MAFFinder* [OMG97], que le proveerán a un cliente MAF[42] de los métodos para crear, recibir, suspender y terminar agentes así como para registrar agentes, lugares y sistemas de agentes respectivamente. De esta forma el cliente MAF podrá acceder a métodos para crear agentes móviles y posteriormente programar las tareas que éste habrá de desempeñar. Esto se puede apreciar en la siguiente figura:

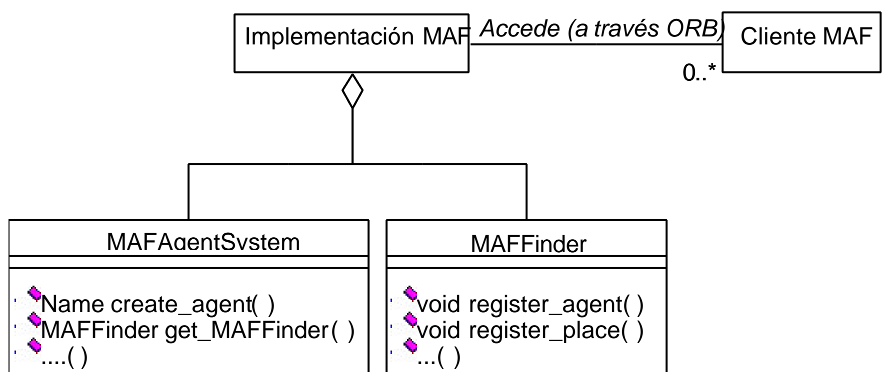
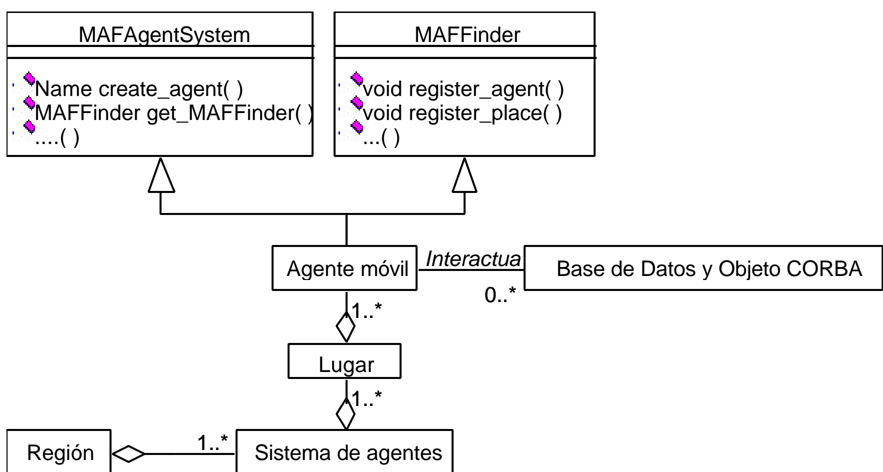


Figura 9.5.1 – La Facilidad de Agentes Móviles de CORBA

En un sistema CORBA los agentes, las clases y los objetos interactúan a un mismo nivel, ya que en principio para crear un agente un cliente MAF (objeto) llama el método *create_agent()* de la interfaz *MAFAgentSystem*.



[42] Que puede ser cualquier objeto o aplicación CORBA

Figura 9.5.2 – Interacción de Agentes Móviles en CORBA

Una vez creado el agente hereda métodos de las interfaces MAF que le permiten acceder a cualquier clase de objeto o base de datos que le sea permitido, o bien comunicarse con otros agentes como se aprecia en la figura 9.5.2.

Bajo este esquema se puede apreciar fácilmente que un agente no será más que un objeto CORBA con capacidades de migración y que será autenticado por la misma MAF, que a través de algunos métodos de la interfaz *MAFAgentSystem* actúa como gestor de seguridad al momento que un agente quiere trasladarse de un sistema a otro.

De igual forma la MAF proporciona información sobre otros sistemas de agentes compatibles y no compatibles a uno especifico, con la finalidad de que un agente pueda saber si puede ser recibido por otro sistema de agentes, o bien saber cual es el sistema de agentes compatible que se encuentra más cerca de él.

Capítulo X

REQUERIMIENTOS DE UNA INFRAESTRUCTURA PARA EL DESARROLLO DE AGENTES MÓVILES Y LA NECESIDAD DE UNA ARQUITECTURA DE SEGURIDAD INTEGRAL.

En ocasiones es difícil o casi imposible aprovechar todas las ventajas que nos ofrece la tecnología de agentes móviles debido a la carencia de una infraestructura apropiada que nos permita resolver los problemas relacionados con conectividad, seguridad, localización, envío de mensajes, etc.

Además dicha infraestructura debe proporcionar diversas herramientas que faciliten nuestro trabajo y que permita realizar tareas complejas de forma transparente al usuario final.

En el capítulo VIII indiqué cuales eran las carencias de los sistemas de agentes móviles actuales, en este capítulo explicaré los requerimientos básicos para una infraestructura de desarrollo, o más bien conocida, como sistema de agentes móviles. De igual forma mencionaré algunas de las herramientas que los sistemas de agentes móviles deben integrar en sus entornos de desarrollo.

Al analizar los requerimientos observamos que la peor carencia de los sistemas actuales es la más grande necesidad, es decir carecen de una arquitectura de seguridad integral que permita que los usuarios emplear la tecnología de agentes sin ningún temor o prejuicio de entidades maliciosas. La segunda parte del capítulo la dedico a justificar la necesidad de una arquitectura de seguridad integral.

10.1 REQUERIMIENTOS DE UN SISTEMA DE AGENTES MÓVILES

Actualmente no existe una infraestructura para el desarrollo de agentes móviles que ofrezca una solución completa a todas las necesidades de la tecnología, sin embargo la plataforma que ofrezca un entorno de desarrollo integral deberá tener ciertas características que le permitan solucionar los requerimientos que exigen la gran gama de aplicaciones que se pueden hacer con agentes móviles. Señalaré los más importantes.

10.1.1 Características deseables del lenguaje de programación de agentes

El código del agente en principio podría ser programado en cualquier tipo de lenguaje, sin embargo existe consideraciones muy importantes que nos hacen seleccionar solo un pequeño grupo de ellos como se señala en [LING96].

La primera consideración y quizá la más importante es que el código del agente debe ser ejecutado idénticamente en cada servidor que él visita, es decir, el código del agente debe ser 100% portable.

Respecto a este punto los compiladores tradicionales (como C por ejemplo) tienen una gran desventaja, ya que ellos normalmente compilan a un lenguaje de máquina que es específico a cada arquitectura y para ejecutar un agente en otro servidor se tendría que enviar su código fuente, lo cual no resulta factible.

Evidentemente los lenguajes interpretados directamente como TCL[OUST94] y Perl[WALL90] o compilados a un lenguaje intermedio que es portable como Java[GOSL95] pueden transportar y ejecutar fácilmente agentes sin necesidad de recompilación.

De esta manera los interpretes portables se convierten en el lenguaje apropiado para programar agentes móviles a pesar de la desventaja ya conocida de los interpretes: el incremento de la velocidad al ejecutar el código.

El lenguaje de desarrollo deberá ser orientado a objetos para que se puedan crear aplicaciones modulares y de gran tamaño que tomen ventaja de todas las bondades de la tecnología y para que los agentes puedan ser tratados como objetos independientes.

De igual forma deberá tener soporte para la persistencia de objetos, de tal forma que permita conservar el estado de las variables de un agente una vez que su ejecución es suspendida y posteriormente reanudada en otro servidor.

Concluyendo podríamos decir que el lenguaje ideal para una infraestructura de desarrollo de agentes será un interprete portable (a la mayor cantidad de plataformas posibles) orientado a objetos y que ofrezca persistencia.

10.1.2 Seguridad

Será el factor principal para que un sistema de agentes pueda ser usado en aplicaciones reales ya que la tecnología no progresará si no se ofrece un esquema de seguridad apropiado en el que se puedan proteger los recursos del servidor y a los agentes.

El sistema deberá proporcionar los mecanismos de protección que garanticen integralmente:

- Protección de la máquina o servidor de agentes contra ataques de agentes o de servidores maliciosos.
- Protección de la transmisión de información contra grupos terceros.
- Protección de los agentes contra servidores maliciosos.

Para lograr estos objetivos es necesario:

- *Autenticación de autoridades mediante firmas digitales:* de esta forma el servidor podrá decidir y asegurarse a que agentes les permite el acceso. También podrá autentificar a los sistemas de agentes para decidir o no intercambiar agentes con determinadas entidades.
- *Encriptación de agentes y mensajes al ser transmitidos:* es necesario establecer un canal de transmisión codificado con lo que se evita la interferencia o el robo de información dentro de la red por parte de grupos terceros.
- *Bases de datos de privilegios:* en donde el servidor pueda especificar todas los privilegios que le otorga a una autoridad específica o bien al servidor de una dirección determinada. Todos los agentes que pertenezcan a dicha autoridad o bien que provengan de una dirección especifica gozaran de los mismos privilegios.
- *Un lenguaje simple de especificación de privilegios:* que permita especificar la base de datos de privilegios para lograr permisos combinados o específicos para una entidad o dirección específica.
- *Asignaciones:* cada agente deberá llevar consigo una estructura de datos que defina la cantidad de recursos que podrá consumir en servidores remotos. Los servidores deberán de encargarse de supervisar el consumo de estos recursos.
- *Protección de los datos y el código del agente:* deberá ofrecer mecanismos para que el código o los datos de los agentes no sean alterados por servidores maliciosos.
- *Cuenta de administrador:* el acceso a estas bases de datos así como a otras opciones de seguridad deberán estar restringidas a un usuario que sea el administrador del sistema de agentes, por lo que deberá haber cuentas múltiples para las necesidades de cada usuario.

10.1.3 Servicios de Agentes

La infraestructura deberá ofrecer a parte del servicio de creación cuando menos tres servicios básicos que a continuación se señalan. De esta forma los agentes podrán implementar aplicaciones más complejas que requieran de interacción y coordinación entre ellos.

10.1.3.1 Movilidad

La infraestructura deberá incluir una instrucción *move_to(host_remoto)* o librería para ofrecer movilidad total. Es decir, el agente podrá enviarse libremente a todas las direcciones que hayan sido definidas en su itinerario. Además deberá considerar y resolver los problemas que se originan al considerar servidores instalados en ordenadores portátiles o bien servidores dentro de un corta fuegos.

Los agentes al viajar pueden tener problemas en alcanzar sus servidores destinos, por ejemplo aquellos que estén temporalmente desconectados de la red. Estos servidores pueden usar colas privadas (llamadas almacenes de agentes[43]), que están localizadas en servidores remotos, en donde los agentes que vayan llegando puedan ser encolados y más tarde recuperados por éstos servidores cuando ellos vuelvan a estar una vez más disponibles o se hayan conectado nuevamente a la red.

Un almacén de agentes permite que los agentes sean compartidos ya que podrían ser recuperados de más de un servidor si se deseara. Proveen una solución segura para la aceptación de agentes dentro de un corta fuegos o dominios de seguridad.

10.1.3.2 Servicio de localización

El servidor deberá proporcionar un servicio de búsqueda y localización de agentes, no sólo de los lugares o agentes locales sino también en servidores remotos, este servicio es indispensable para que posteriormente los agentes puedan interaccionar con los agentes de otros sitios.

La localización de agentes en la red facilita la comunicación remota entre agentes y la gestión remota del agente. Por ejemplo, el propietario de un agente remoto podría enviarle un mensaje de control (eliminarlo) o personalizar su comportamiento enviándole otro itinerario. Otro ejemplo puede ser el caso en el que se envían múltiples agentes a distintos servidores con el objeto de ejecutar tareas en paralelo los cuales pueden requerir comunicarse con el fin de intercambiar resultados parciales o sincronizarse.

El servicio de localización será básico para el servicio de comunicación y para el monitor del sistema de agentes los cuales serán descritos posteriormente.

Existen tres esquemas básicos para la localización de agentes como se aprecia en la figura 10.1.3.2.1, que son:

- *Secuencias de acceso:* un agente es localizado siguiendo los rastros de información que va dejando al visitar cada servidor. Es decir, cada servidor guarda una referencia de todos los agentes que le visitan así como el destino remoto al que partió.
- *Fuerza bruta:* un agente es localizado al buscar por él en múltiples servidores remotos. La búsqueda puede ser hecha en paralelo o en secuencia.

[43] De los términos en ingles *Agent box*

- *Registro en un servidor de búsqueda:* un agente puede actualizar su localización en un servidor de nombres establecido que les permita a los agentes buscar, registrarse o borrarse. De esta manera los agentes y servidores pueden localizar a un agente en especifico.

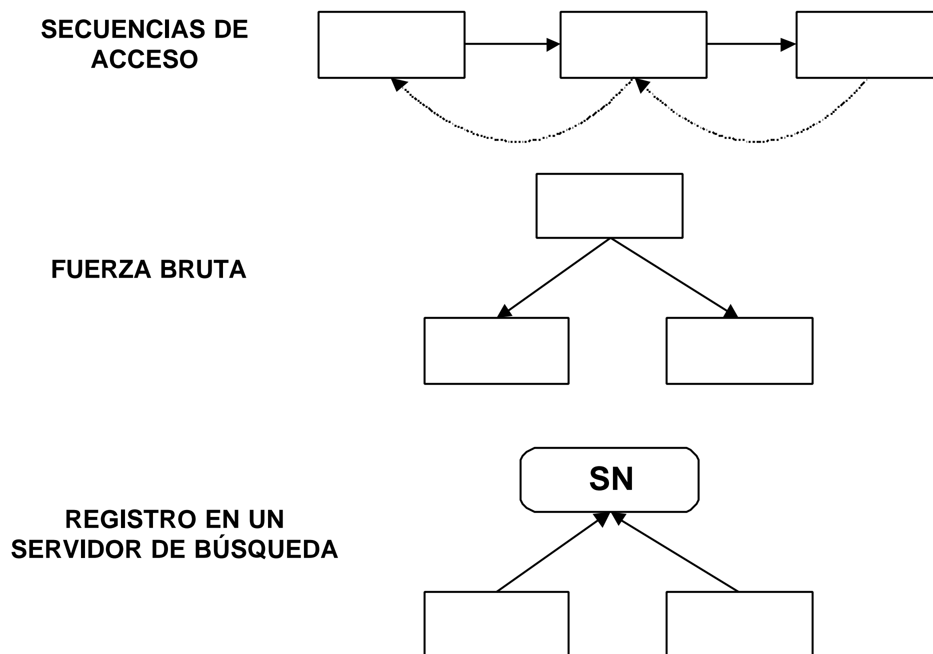


Figura 10.1.3.2.1 – Esquemas para la localización de agentes

10.1.3.3 Comunicación

El sistema permitirá la transferencia de mensajes síncronos y asíncronos no sólo entre agentes locales sino también entre agentes remotos. La comunicación les permitirá realizar una interacción entre ellos para lograr realizar tareas en conjunto.

De esta forma por ejemplo, se podría implementar una búsqueda en paralelo, enviando a varios agentes a la vez en busca de una información deseada, cuando un agente encuentre la información requerida o parte de ella, podría avisar al resto a través de un mensaje que suspendan su búsqueda, o bien que continúen colaborando enviándoles la información a buscar que falta y rutas alternativas de búsqueda a los otros agentes.

Existen dos esquemas básicos para la entrega de mensajes:

- *Localizar y transferir:* el agente es localizado y enseguida le es enviado el mensaje directamente a él, es este caso la transferencia del mensaje implica dos fases.
- *Reenvío secuencial:* en este caso la localización y la entrega del mensaje se hacen en la misma fase. El mensaje es redireccionado usando las secuencias de acceso de los servidores por donde fue pasando.

El servidor debe cuidar que no se haga un uso excesivo de este servicio para que no se sature la red de mensajes, ya que va en contra de la filosofía de los agentes móviles.

10.1.4 Interoperabilidad con otros sistemas de agentes

Deberá definir un modelo de agente estándar cuya especificación pueda ser autentificada y transportada a otros sistemas de agentes. O bien deberá de encargarse de hacer puentes a los sistemas de agentes más populares con el fin de lograr interoperabilidad.

La especificación deberá estar fundamentada en el estándar MASIF para facilitar la también la interoperabilidad con los ORB's CORBA. Asimismo deberá considerar los aspectos de seguridad vistos anteriormente para que al hacer los puentes con otros sistemas se puedan llevar a cabo todas las validaciones necesarias, ya que deberá reconocer agentes de otros sistemas para que puedan navegar dentro del sistema de agentes local.

10.1.5 Integración al WEB

Hoy en día en WEB es una infraestructura ubicua por lo que todas las nuevas tecnologías de software se están integrando a él. Los navegadores y las páginas WEB extendidas mundialmente se convierten en herramientas perfectas para la comunicación con agentes móviles ya que los clientes no necesitarían instalar ningún software propietario o familiarizarse con una nueva interfaz.

10.1.5.1 Integración a los navegadores

El sistema de agentes deberá ser diseñado para que los agentes se puedan integrar fácilmente a los navegadores, para que los usuarios de Internet puedan usar la tecnología para buscar y recopilar datos. Los sistemas de agentes desarrollados en Java utilizarían la misma máquina virtual que utiliza el navegador para interpretar los applets.

Esta característica proporcionará a los navegadores un estilo de búsquedas asíncronas en donde los usuarios podrán dejar el ordenador por un tiempo y regresar más tarde para ver los resultados que sea han obtenido.

10.1.5.2 Agentes dentro de applets

Un applet sólo puede establecer comunicación con su servidor original, los agentes que sean enviados por él son transferidos transparentemente a un servidor de encaminamiento especial que se encuentra en el servidor original, el cual a través de dicho servidor se encarga de enviar a los agentes a su servidor destino.

A diferencia de los usuarios exclusivos de agentes móviles, los clientes no tendrían que instalar ningún tipo de software adicional con el fin de ejecutar los agentes dentro de los

applets. Además, los agentes que son descargados con sus applets y ejecutados dentro de sus navegadores pueden ser autentificados como parte de la autenticación del applet y controlado por el gestor de seguridad de applets. El sistema de agentes de *Aglets* ya ha empezado a indagar en este tema [ARID98b].

10.1.5.3 Páginas amarillas

Los agentes deberán de ser capaces no sólo de buscar a otros agentes o lugares especificando el nombre deseado, sino que también serán capaces de localizar un servidor que contenga el tipo de información que desea.

Para lograrlo cada servidor deberá enviar una lista a un servidor central de páginas amarillas del tipo de información al que se puede acceder en dicho servidor, de tal forma que cuando un agente busque un servicio específico, tendrá que contactar con el servidor de páginas amarillas para que éste le dé la dirección específica del sitio en donde puede encontrar dicha información.

10.1.5 Interfaces gráficas y monitores de agentes

Un sistema de agentes deberá contar con interfaces gráficas para facilitar las necesidades de todo tipo de usuarios. Una vez lanzado el agente, se podrá administrar su comportamiento(por ejemplo hacer que regrese o eliminarlo) de forma gráfica.

Para definir el itinerario de agentes de podrá hacer de forma visual o mediante interfaces gráficas que faciliten las tares. Por otra parte los agentes una vez creados, deben ser capaces de proporcionar una forma de interacción gráfica con el usuario que le facilite definir el tipo de información que requiere o la tarea que desea ejecutar.

De igual forma deberá proporcionar un servicio de monitorización en dónde se muestren todos los agentes activos en el sistema indicando para cada uno de ellos su servidor de origen. Así mismo mostrarán un monitor remoto que permita visualizar la localización actual de todos los agentes que fueron creados en dicho servidor.

10.1.6 Herramientas de desarrollo

Finalmente los servidores requieren otro tipo de herramientas que le faciliten a los programadores de agentes el desarrollo de sus aplicaciones.

10.1.6.1 Entornos visuales de desarrollo

El sistema deberá de proporcionar un entorno de desarrollo visual que permita realizar aplicaciones basadas es agentes móviles por cualquier tipo de usuario, no solo por aquellos que tengan conocimientos de programación.

Es decir, deberán de integrar entornos en donde con sólo cortar y pegar o arrastrar y soltar se puedan definir las acciones que ejecutará el agente así como el itinerario que va a seguir.

10.1.6.2 Un depurador y un gestor de excepciones

En principio la primera necesidad para los programadores será un depurador que les permita encontrar más fácilmente los fallos en los programas. Este depurador deberá simular el envío y ejecución de las tareas del agente en cada uno de los destinos que habría de visitar en su itinerario para poder deducir en donde se encuentra la falla. Es evidente que un depurador de este tipo de programas no es fácil de implementar.

Aunado a él se requiere un gestor de excepciones que envíe al sistema de agentes origen el tipo de excepción que un agente remoto haya producido. Es decir, la infraestructura de agentes deberá de incorporar un gestor de excepciones que le permita enviar al servidor de procedencia de un agente, la excepción que éste haya generado.

10.2 NECESIDAD DE UNA ARQUITECTURA DE SEGURIDAD INTEGRAL

Es importante señalar que muchos sistemas hoy en día ofrecen diversas herramientas de las que he descrito en este capítulo (por ejemplo los Aglets [LANG96] y Ara [PEIN97a]), no obstante presentan muchas deficiencias en sus esquemas de seguridad.

Debemos de considerar que por muy buena que sea una aplicación de agentes móviles, ésta no será usada en el mundo real si la infraestructura sobre la cual fue desarrollada no ofrece una garantía total de seguridad. De hecho, la aplicación no podrá tener éxito si no incluye un modelo de seguridad versátil y eficiente que tendrá que ser soportado por el sistema de agentes en el que fue desarrollada.

Existen una serie de carencias y problemas en los esquemas de seguridad actuales que originan la necesidad de una arquitectura de seguridad integral. La mayor parte de los sistemas de agentes actuales se han orientado al desarrollo de herramientas visuales o de programación, pero no le han dado la importancia necesaria a la seguridad por lo que ningún sistema de agentes cuenta con una arquitectura de seguridad integral apropiada que evite todos los ataques que un sistema de agentes puede tener y que garantice integridad a todos los elementos que interaccionan: al servidor de agentes, a los agentes móviles y a las comunicaciones que se efectúan entre ellos.

Solo cuando exista un sistema que ofrezca dicha garantía podrá ser utilizado para desarrollar aplicaciones que se utilicen en le mundo real. Algunas de las razones o carencias que dan origen a la necesidad de una arquitectura de seguridad integral las describo a continuación:

10.2.1 La falta de protección total del servidor de agentes

Nadie desea exponer un ordenador a los daños que pueden causar agentes provenientes de cualquier parte del mundo, en donde en muchos casos su código es desconocido. La protección de servidor es fundamental para el buen funcionamiento del sistema de agentes ya que él es el encargado de asignar y administrar los recursos. Sin embargo existe una gran gama de ataques a los que está expuesto.

10.2.1.1 El problema de falsificación de agentes y servidores

La falsificación ocurre cuando una entidad quiere hacerse pasar por otra con el fin de obtener mayores privilegios o bien acceder a información que en condiciones normales no le estaría permitido. La falsificación puede ocurrir de varias maneras, por ejemplo un servidor de agentes puede intentar hacerse pasar por otro servidor de autoridad distinta, o bien un agente puede decir que pertenece a una autoridad que no es la suya con el fin de que le sean otorgados más privilegios.

Es necesario que un sistema de agentes sea capaz de autentificar tanto a los servidores como a los agentes con los que interacciona para evitar cualquier peligro de falsificación[44]. Para lograrlo puede hacer uso de técnicas criptográficas avanzadas, por ejemplo firmas digitales.

Cada sistema de agentes y cada agente deben pertenecer a una autoridad (entidad) reconocida por el sistema local. A su vez, el sistema de agentes debe proporcionar bases de datos para almacenar las autoridades válidas y diversos mecanismos que faciliten la autenticación de las autoridades del sistema de agentes remoto y de sus agentes.

10.2.1.2 Los esquemas inadecuados para la asignación de privilegios

El problema más típico que se presenta es que un agente accede a información o a recursos del sistema a los cuales no le esta permitido su acceso, produciendo con ello el típico ataque de *acceso ilegal*. La principal causa de este problema es la ejecución de los agentes dentro de entornos de ejecución (sistema de agentes) inseguros.

[44] Ya que la asignación de permisos normalmente se hace normalmente en función de la autoridad que se autentifica.

Para solucionar este tipo de ataques los primeros sistemas de agentes basados en Java (los aglets por ejemplo [LANG96]) decidieron crear dos grandes grupos de agentes, los locales llamados "confiables" y los remotos llamados "no confiables". Los confiables tenían privilegios sobre todos los recursos del sistema y los no confiables no podían acceder a ninguna parte del sistema de ficheros ni a ningún otro recurso, o bien, si se les daba algún privilegio al ser tratados como un grupo atómico, el permiso se otorgaba a todos los agentes remotos.

Este esquema de protección el cual se basa en el modelo de seguridad del JDK 1.1.6 y que es usado por la mayoría de los sistemas de agentes móviles desarrollados en Java resulta inapropiado y poco versátil básicamente por las siguientes razones:

1. No es posible asignar permisos individuales a cada autoridad que sea propietaria de un grupo de agentes, por lo que sí se concede un permiso es asignado a todos los agentes remotos por igual.
2. A la hora de asignar permisos sólo es posible definir los privilegios de acceso, (como lectura, escritura o ejecución) pero es imposible definir sobre que parte del sistema de ficheros (carpeta o archivo) es concedido el privilegio. Es decir los privilegios que se otorgan se aplican sobre todo el sistema de ficheros.

El sistema de seguridad de un sistema de agentes debe permitir la definición de permisos particulares a cada autoridad local y remota que se reconozca en el sistema de tal forma que tanto los agentes remotos como los locales puedan gozar de privilegios diferentes. Adicionalmente, deberá permitir que los privilegios de acceso sean concedidos de forma específica a algún recurso del sistema. Por ejemplo, sólo a un directorio o fichero específico dentro de todo el sistema de ficheros.

10.2.1.3 La mala protección contra el consumo de recursos del sistema

Otro ataque que se puede presentar es el típico "*caballo de Troya*", esto es, un agente que es ejecutado por un usuario legitimo pero que realiza algo diferente que lo que el sistema de agentes esperaba que hiciera, este tipo de agentes pueden causar daños a los demás agentes en el sistema o bien a los recursos del sistema.

Los sistemas de agentes igualmente son muy vulnerables al ataque de *agotamiento de recursos* también conocido como ataque DOS[45]. Este ataque consiste es que un agente usa deliberadamente un recurso (por ejemplo memoria) de tal forma que dicho servicio sea negado a los demás agentes una vez que se haya agotado dicho recurso.

Es importante considerar que cualquier agente fácilmente puede realizar este último ataque para agotar los recursos de un sistema, basta con enviar un pequeño e "inofensivo" agente que la única función que se especifique en su código sea la de crear un agente hijo pero dentro de un ciclo *for* infinito. La creación indefinida de agentes hijos corrompería el sistema.

[45] De sus siglas en inglés Denial of service (negado de servicio)

Debido a estos ataques, es necesario que el sistema de seguridad de un sistema de agentes no sólo controle el número de recursos que puede consumir cada agente sino la cantidad de recursos que puede consumir cada autoridad. Esto es, si sólo se controla la cantidad de recursos que puede consumir un agente pero no su autoridad, dicha autoridad puede utilizar el mismo ciclo *for* para enviar indefinidamente agentes a un servidor remoto en donde la única función de cada agente enviado será la de llegar y crear un hijo. Después de un tiempo el sistema remoto será igualmente corrompido.

Para evitar estos ataques es necesario que cada agente cuente con una asignación[46] que determine los recursos que él puede consumir, y que el sistema de seguridad del servidor de agentes remoto cuantifique (basándose en la asignación) las cantidades consumidas por los agentes de cada autoridad. El servidor de agentes se asegurará que los agentes no sobrepasen la cantidad de recursos que le han sido asignados a su autoridad. Cuando la cantidad de recursos disponibles para esa autoridad sea agotada no se deberán recibir más agentes de dicha autoridad.

10.2.1.4 La carencia de versatilidad de las políticas de seguridad en tiempo de ejecución

Si bien es cierto que los sistemas de agentes móviles actuales no cuentan con modelos de seguridad integrales que eviten todos los ataques que estoy describiendo en esta sección, menos aún contarán con un sistema de seguridad versátil que permita en casos necesarios modificar la política de seguridad en vigor.

Existen principalmente dos casos en que puede ser necesaria la modificación de la política de seguridad. El primero de ellos se deriva del trabajo en grupo de agentes móviles, como describí anteriormente uno de los beneficios fundamentales de la tecnología es el desarrollo de aplicaciones en paralelo en donde un agente probablemente va a crear un fichero que va a ser usado por otro posteriormente, pero ¿qué sucede con ese fichero que se ha creado durante la ejecución?, ¿Queda protegido de tal forma que no pueda ser accedido por otros agentes o entidades no deseadas?. La respuesta es NO.

El segundo es cuando el sistema detecta que se le están agotando los recursos fundamentales del sistema como RAM y HD, o bien que la red esta sobre saturada, entonces es necesario que la política en vigor sea remplazada temporalmente con una política de emergencia (que será muy restrictiva) hasta que algunos agentes migren y se recuperen dichos recursos para que el sistema vuelva a ser estable.

De acuerdo a lo anterior, un sistema de seguridad versátil debe permitir que un agente móvil incluya los ficheros que genere en tiempo de ejecución a la política de seguridad en vigor indicando que privilegios de acceso otorga (leer, escribir, ejecutar) y a que autoridades les concede dichos permisos. De igual forma, el sistema de seguridad debe estar preparado para utilizar una política de seguridad de emergencia cuando sea necesario.

[46] Un objeto en donde se cuantifican la cantidad de recursos que el agente puede consumir en un sistema remoto.

10.2.2 Carencia de seguridad en la transmisión de agentes

Actualmente muy pocos sistemas de agentes se han preocupado por proteger a los agentes y a la información que transmiten, quizá se han enfocado en brindar protección al servidor y han descuidado este punto.

Durante la transmisión de agentes móviles los sistemas de agentes pueden ser víctimas de ataques pasivos[47] por parte de grupos terceros que deseen acceder a la información que llevan los agentes consigo o a los mensajes que se envían entre ellos.

10.2.2.1 Espionaje de la información

El ataque más probable y difícil de detectar de este género es el *espionaje de agentes* y sus datos. Consiste en usar un programa llamado "monitor de comunicación". Dicho programa permanece observando la información que es intercambiada entre los sistemas de agentes con el fin de capturar agentes para extraer información (sin colaboración y permiso del agente) de importancia para el grupo tercero que realiza el ataque. Es difícil de detectar porque no se altera de ninguna forma al agente, el cual puede continuar con su ejecución. Este se representa gráficamente en la siguiente figura:

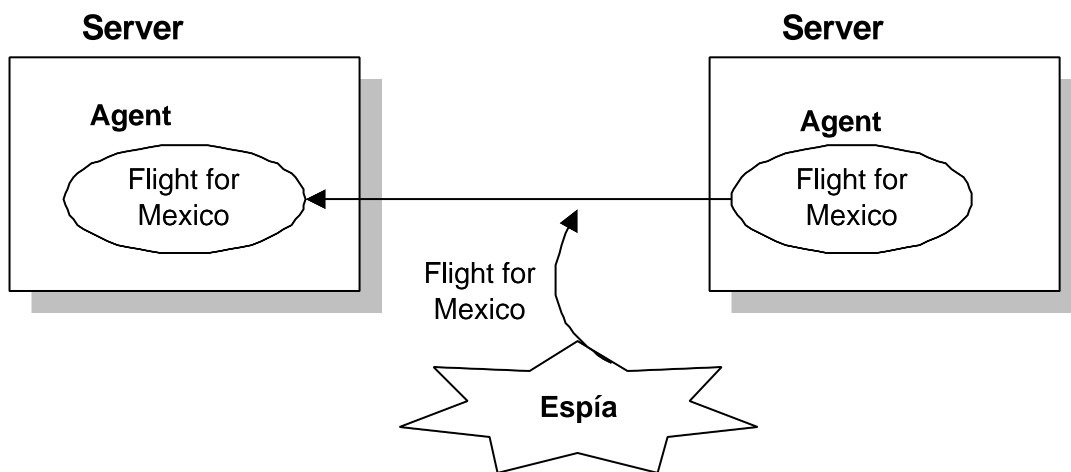


Figura 10.2.2.1.1 – Espionaje de Información al transmitir datos

10.2.2.2 Poca fiabilidad en la transmisión de mensajes

Una variante del ataque anterior es el ataque de *interceptación y sustitución* de mensajes. Este ataque usa un monitor de comunicación inteligente capaz de extraer mensajes con el fin de analizar la información recibida y luego sustituirla para enviar un mensaje alterado que beneficie al atacante. Por ejemplo supongamos que un agente va a recibir una dirección de donde comprar un producto, el atacante intercepta el mensaje y modifica la dirección para que el agente vaya a comprar a su tienda.

[47] Aquellos ataques que no modifican en nada la estructura y los datos del agente.

Es necesario que un sistema de agentes use técnicas criptográficas avanzadas o bien un protocolo de transferencia de información encriptada que permita evitar los dos tipos de ataques que se han mencionado anteriormente.

Las técnicas criptográficas previenen de que los espías sean capaces de entender la información que roban (ya que no cuentan con la clave necesaria para decodificarla) y los protocolos de transferencia como SSL crean un canal de transmisión codificado de donde no se puede robar ningún tipo de información que no este codificada y por tanto incomprensible para las terceras partes.

10.2.3 Falta de protección del agente y sus datos contra ataques de un servidor malicioso

Si la mayor parte de los sistemas de agentes actuales no cuentan con mecanismos de protección para resolver los ataques descritos anteriormente, menos aún tendrán una forma de protegerse contra el problema más difícil de solucionar en los sistemas de agentes móviles que es la protección del agente y sus datos contra atraques del servidor.

Durante el recorrido de un agente móvil, éste puede ser atacado por un servidor malicioso que altere su código o sus datos. De igual forma el servidor puede interceptar al agente y enviar una replica para obtener resultados idénticos que podrá mal emplear posteriormente.

10.2.3.1 El problema de alteración código y datos

Cuando un servidor recibe un agente, éste crea una instancia de la clase que dio origen al agente, lee su estado de datos serializado y crea un hilo de ejecución para ejecutarlo. Como puede suponerse, desde un principio el servidor tiene completo control sobre el agente y tiene acceso total a su código y datos.

Un servidor malicioso puede alterar el código de un agente móvil con el fin de modificar su comportamiento y de que realice tareas que favorezcan sus perversos intereses o bien que ejecute tareas destructivas en otros servidores. El cual puede apreciarse en la figura 10.2.3.1.1.

Para evitar este ataque de *alteración de código* es indispensable que el servidor de agentes utilice técnicas criptográficas para protegerles. Las firmas digitales permiten firmar información que va a ser transmitida de tal forma que sí se modifica la información firmada durante su trayecto la firma que va adjunta a la información ya no es válida con los que se detecte inmediatamente que la información ha sido alterada.

Los agentes antes de ser enviados pueden ser firmados por la autoridad a la que pertenecen, cuando lleguen a un nuevo servidor estos validarán su firma adjunta. Si el código ha sido modificado la firma no será valida y el agente será rechazado, de esta forma no se les permitiría el acceso a los agentes cuyo código ha sido alterado.

El ataque de *alteración de datos* resulta ser mucho más complejo debido a que los datos del agente van cambiando durante su itinerario. Los datos del agente podrían ser firmados por la autoridad de cada sistema de agentes por el que pase, no obstante esto sería mas que nada con la finalidad de adjudicar responsabilidad sobre los servidores, es decir, la firma sería como una garantía de que ellos no han modificado los datos.

Sin embargo, podría darse el caso de que un servidor firme a pesar de que haya alterado los datos para poder evitar este problema es necesario que el servidor de agentes utilice técnicas criptográficas avanzadas o códigos de autenticación de resultados parciales (PRAC) que le permitan al agente protegerse contra dichos ataques o bien cuando menos detectar de que sus datos han sido modificados.

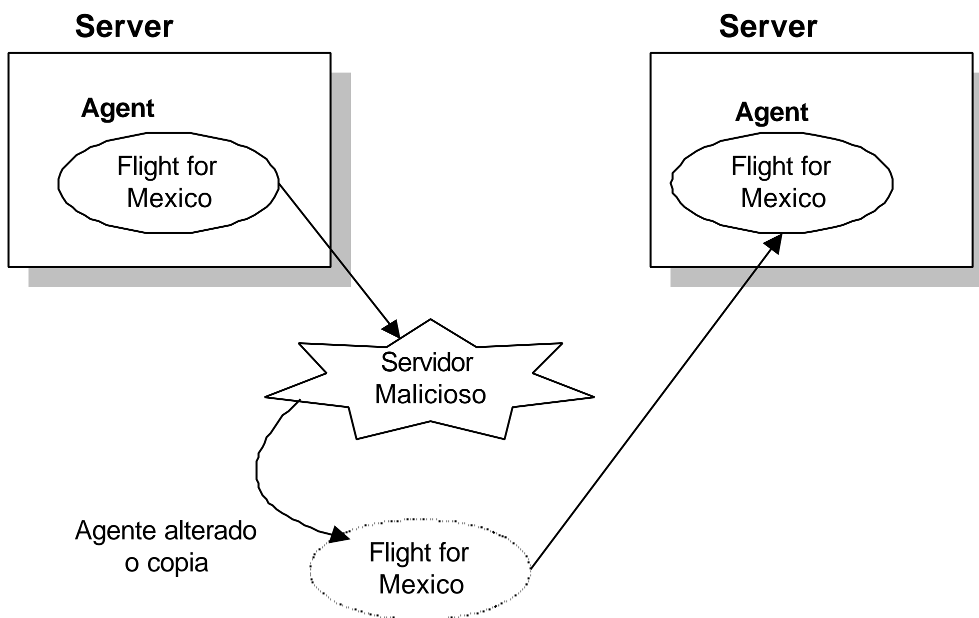


Figura 10.2.3.1.1 – Alteración y Replicas

10.2.3.2 El problema de replicas

Un último problema al que se enfrentan los sistemas de agentes es el ataque de *replicas,* que consiste en que un servidor captura una copia de un agente enviado previamente y es retransmitido después con los propósitos particulares del servidor y no con los fines para los que fue creado. De esta forma el servidor malicioso podría obtener con la copia del agente resultados similares a los del agente original.

Para que la copia del agente regrese al servidor malicioso quizá será necesario modificar su comportamiento[48] con lo cual el ataque podría prevenirse al igual que en el primer caso con el uso de firmas digitales. Si la finalidad de la interceptación del agente es la de enviar una copia a otro servidor con fines destructivos o a un servidor que se encargue de analizar su contenido la protección de dicho ataque resulta más difícil. Podrían emplearse PRAC para detectar alguna alteración en los datos del agente.

[48] Esto dependerá de cómo el agente haya definido su itinerario.

Es necesario que el sistema de agentes al igual que para todos los ataques descritos anteriormente ofrezca un mecanismo de protección apropiado que garantice la integridad del agente durante su recorrido.

PARTE III .- ARQUITECTURA DE SEGURIDAD INTEGRAL PARA SISTEMAS DE AGENTES MOVILES

Capítulo XI
SAHARA: ARQUITECTURA DE SEGURIDAD INTEGRAL PARA SISTEMAS DE AGENTES MÓVILES

De acuerdo al análisis desarrollado en este trabajo un sistema de agentes móviles para poder ser utilizados en aplicaciones reales requiere necesariamente dos características fundamentales como lo son:

Portabilidad: lo que permitirá a los agentes móviles moverse en redes heterogéneas.
Seguridad: lo que permitirá que los usuarios confíen completamente en la tecnología y se empiecen a realizar todo tipo de aplicaciones que aprovechen todos los beneficios de la tecnología.

De igual forma en este trabajo he mostrado como la evolución de los agentes móviles tiende hacia Java. La aparición de la tecnología revolucionó la investigación en agentes móviles ya que satisfacía totalmente[49] el requerimiento de portabilidad. Un sistema de agentes basado en Java así como los agentes creados en él podrían navegar libremente en casi todas las plataformas actuales del mercado, debido a esto todos los sistemas de agentes móviles que existían[50] han incorporado al lenguaje Java como una plataforma más de desarrollo.

De esta forma podemos concluir que la plataforma dominante para el desarrollo de agentes móviles es Java y que sólo tiene solucionado el problema de la portabilidad, ya que actualmente no se goza de una arquitectura de seguridad bien diseñada que satisfaga todos los requerimientos de una aplicación real y garantice una total confianza.

La arquitectura de seguridad SAHARA ofrece un modelo de seguridad que puede ser implementado por cualquier sistema de agentes móviles. SAHARA esta basada en un análisis concienzudo de todos los puntos vulnerables que un sistema de agentes móviles puede tener y ofrece una solución satisfactoria y total a casi todos ellos.

[49] Ya que actualmente la máquina virtual de Java se encuentra disponible en todas las plataformas más importantes del mercado.
[50] Como D'Agents y Ara

SAHARA ha sido concebida para ser implementada sobre cualquier plataforma orientada a objetos, su arquitectura modular y bien diseñada es lo suficientemente versátil para ser implementada por cualquier sistema de agentes actual. No obstante la implantación en un sistema de agentes basado en Java que utilice el JDK 1.2 facilitaría algunas tareas de implementación, ya que se utilizarían algunas herramientas de seguridad que proporciona el JDK en dicha versión, no obstante el modelo pueden ser implementado con versiones anteriores del JDK o con cualquier plataforma de desarrollo que soporte la orientación a objetos.

11.1 INTRODUCCION A LA SEGURIDAD EN LOS SISTEMAS DE AGENTES MÓVILES

Un agente móvil es un programa que se mueve de un ordenador a otro y se ejecuta en cada una de ellos. Debemos suponer que ni el agente ni los ordenadores son necesariamente confiables. El agente puede tratar de acceder o destruir información privilegiada, o bien puede consumir una gran cantidad de recursos que terminen agotando los recursos del servidor.

Por su parte el ordenador servidor podría tratar de extraer información secreta del agente, cambiar su comportamiento al afectar el código o los datos, o bien modificar la información que el agente recopila si esto le produce algún beneficio al servidor.

Un sistema de agentes móviles que no sea capaz de detectar y prevenir tales acciones nunca podrá ser usado en aplicaciones reales. En un entorno de red abierta, los ataques intencionales sobre el servidor y los agentes comenzarán tan pronto como el sistema sea iniciado, y aún en un entorno de red cerrado con usuarios confiables existe el peligro de agentes mal programados que puedan causar un daño significativo accidentalmente.

La carencia de una arquitectura de seguridad confiable es el factor principal por lo que la tecnología de agentes no se ha difundido ampliamente debido a ello la seguridad es quizá el aspecto más crítico en un sistema de agentes móviles y puede dividirse en seis problemas interrelacionados que se originan de todas las interacciones posibles entre las entidades de un sistema de agentes [HOHL97]: interacción entre agentes, entre servidores o sistemas de agentes y agentes, entre servidores e interacción entre sistemas de agentes y terceros grupos no autorizados.

Así pues las áreas que se deben cuidar son:

Protección de la máquina o servidor de agentes contra ataques de agentes: la máquina debe ser capaz de autenticar al propietario del agente para evitar falsificaciones y de asignarle límites a sus recursos basados en esta autenticación. De igual forma debe prevenir la violación de estos límites para tener una integridad total. Para evitar ataques de *acceso ilegal* y de *agotamiento de recursos*, los límites en los recursos deben incluir privilegios de acceso (de lectura y escritura) así como las cantidades máximas que los agentes podrán consumir de dichos recursos.

Protección del agente contra otros agentes: ningún agente debe ser capaz de interferir con otro agente o bien robarle recursos. De igual forma debe evitarse cualquier tipo de falsificación de tal forma que un agente no logre obtener información privada de otro agente falsificando la identidad de un tercero. Este tipo de protección puede ser visto como un subproblema de la protección de la máquina ya que si un agente no logra sobrepasar los límites de recursos que le han sido establecidos ni logra descifrar los mecanismos de comunicación difícilmente podrá afectar a otro agente.

Protección del agente contra el servidor de agentes: una máquina no debe de ser capaz de manipular el comportamiento o extraer información sensible del agente sin su cooperación, es decir, debe protegerse al agente contra ataques de *replicas* y *alteración*. Desgraciadamente, este problema es el más difícil de resolver ya que la única solución completamente garantizada para resolver este problema es mediante el uso de hardware [WILH98], ya que es casi imposible prevenir que la máquina haga lo que quiera con el agente ya que el agente se encuentra ejecutando sobre la máquina. No obstante existen alternativas para detectar la falsificación o la manipulación de código de tal manera que un agente que ha sido afectado pueda ser detectado para eliminarlo o repararlo en caso de que esto fuera factible.

Protección de un grupo de máquinas contra agentes: un agente podría consumir recursos excesivos en una red aunque sólo consuma unos cuantos recursos en cada máquina. Un ejemplo obvio sería el de un agente que navega en la red indefinidamente o un agente que crea dos hijos en cada ordenador y estos a su vez se duplican cada vez que migren a otro sitio hasta inundar toda la red. El sistema de agentes debe estar preparado para resolver estos problemas.

Protección de la máquina o servidor de agentes contra otras máquinas: al igual que en primer caso, una máquina puede intentar falsificar la identidad de otra para obtener más privilegios al realizar una solicitud a un tercer servidor. Al igual que para el primer caso, la mejor forma de solucionar este problema es la autenticación mediante el uso de firmas digitales y encriptación.

Protección entre servidores de agentes al transmitir información contra terceros no autorizados: tenemos que considerar en esta área la seguridad de comunicación entre dos servidores sobre una red insegura, el espionaje de información así como los diversos ataques ya bien conocidos y mencionados en detalle en [STAL95], [SCHN96] y [CHES97] son los que se deberán de eliminar. La mejor forma es la utilización de un protocolo para transmitir la información que sea completamente confiable.

La mayor parte de estos problemas han sido considerados en la literatura de agentes móviles, sin embargo sólo algunos sistemas presentan soluciones parciales a algunos de ellos, quizá la madurez de los sistemas precursores como D'Agents y ARA les ha permitido que evolucionen ofreciendo arquitecturas de seguridad más completas[51]. Sin embargo las arquitecturas de seguridad para los nuevos sistemas de agentes basados en Java solo ofrecen una solución parcial a los problemas expuestos anteriormente con lo cual no ofrecen una garantía suficiente para que puedan usarse en aplicaciones reales.

SAHARA ofrece una solución para casi todos los problemas mencionados realizando un esquema de protección apropiado y considerando todos los puntos vulnerables de un sistema de agentes. Analizaré en principio todo lo referente a la protección del servidor

[51] No obstante lejos aún de solucionar todos los problemas de seguridad que un sistema de agentes puede sufrir.

de agentes para pasar posteriormente a los medios para proteger al agente, no sin antes definir algunos conceptos que serán de utilidad para comprender la siguiente sección.

11.1.1 Conceptos básicos

Para lograr entender los principios fundamentales sobre los que se basa la arquitectura de SAHARA es necesario comprender algunos de los conceptos básicos:

Autoridad: es una entidad cuya identidad puede ser autentificada por un sistema al que la autoridad esta tratando de acceder. Una entidad puede ser un individuo, una corporación o una empresa y consiste normalmente de un nombre[52] mediante el cual suele ser identificado. En algunos sistemas de agentes al concepto de **autoridad** se le conoce como **principal.**

Certificado: una declaración firmada digitalmente de una entidad, diciendo que la llave[53] pública de alguna otra entidad tiene un valor particular. Si se confía en la entidad que firmo el certificado entonces se tiene confianza en que la llave pública de la otra autoridad en auténtica.

Llave pública: un número asociado con una autoridad particular (un individuo u organización). Una llave pública se pretende dar a conocer a todos aquellos que necesitan tener interacciones con aquella entidad.

Llave privada: un número que sólo es conocido por una entidad particular. Esto es, las llaves privadas siempre son un secreto. Una llave privada siempre esta asociada con una llave pública específica.

Firma digital: una cadena de bits que se calcula a partir de algún dato (el dato que esta siendo firmado) y la llave privada de una entidad. La firma puede ser usada para verificar que los datos vienen de la entidad. Como una firma manuscrita la firma digital tiene muchas características:

- Su autenticidad puede ser verificada mediante un cálculo que usa la llave pública correspondiente que se usa para generar las firmas.
- No puede ser falsificada, ya que la llave privada es una secreto
- Los datos no pueden ser modificados, si lo son, la firma no seguirá siendo válida, ya que perderá su autenticidad.

Encriptación: es el proceso de tomar datos (llamados texto legible) y una llave (una cadena corta) para producir un texto cifrado o encriptado, el cual es un dato sin ningún significado para grupos terceros que no conozcan la llave.

Desencriptación: es el proceso contrario a la encriptación que consiste en tomar el texto cifrado y la llave para producir texto legible.

Algoritmo criptográfico: es usado para asegurar una o más de las siguientes funcionalidades:

- La confidencialidad de los datos.
- La autenticación del emisor de los datos.
- La integridad de los datos enviados.
- La imposibilidad de negar un envío previo.

El algoritmo de firmas digitales proporciona la mayor parte de estas características.

[52] Es posible que contenga algunos otros atributos

[53] Del ingles *key*, en este documento utilizaré el concepto de llave usado en América o el de clave usado en España indistintamente.

Para consultar cualquier otro término refiérase al glosario adjunto o bien en [DAGE96], [SUNa] y [KARJ97].

11.2 ESQUEMA GLOBAL DE SAHARA EN LA INTERACCIÓN ENTRE SISTEMAS DE AGENTES MÓVILES

SAHARA utiliza conceptos del estándar de CORBA para la interoperabilidad de sistemas de agentes (MASIF) que vimos con anterioridad en el capítulo VIII. De igual forma utiliza las técnicas más avanzadas de criptografía en su esquema de protección así como protocolos de transferencia segura en la implementación del protocolo de transferencia de agentes (PTA). Dichas características le permiten resolver los dos problemas básicos de seguridad de los sistemas de agentes utilizando esquemas estándar.

Esta arquitectura esta diseñada para permitir la interoperabilidad entre los sistemas de agentes que la implementen y que posean un mismo interprete. Al cumplir con el PTA especificado por SAHARA (que estará implementado sobre SSL) cualquier sistema de agentes podrá recibir agentes de servidores remotos que utilicen el mismo interprete para programar los agentes[54], sin importar que algunos sistemas de agentes ofrezcan más o menos servicios o tengan algunas otras diferencias en su implementación.

De igual forma, al comunicarse con un sistema de agentes que cumpla la especificación MASIF de CORBA, SAHARA no afecta y permite la comunicación con otros objetos dentro de un ORB CORBA para lograr una interacción total como lo muestra la figura 11.2.1.

Aunado a esto implementa patrones de navegación y añade a sus agentes funciones de verificación que protegen los datos del agente contra ataques de servidores maliciosos con lo que se protegen en gran medida todos los aspectos de seguridad que un sistema de agentes pueda sufrir.

Antes de analizar los diversos aspectos de la arquitectura de seguridad SAHARA, comenzaré por definir los principios fundamentales sobre los que esta edificada así como el modelo global de interacción que sigue para lograr una amplia y confiable interoperabilidad.

11.2.1 Objetivos de la arquitectura

El objetivo principal de esta investigación es crear una arquitectura de seguridad para los sistemas de agentes móviles que garantice una seguridad integral a cualquier sistema de agentes que la implemente para que este pueda ser usado en aplicaciones reales y

[54] El hecho de recibirlos implica la capacidad de poder ejecutarlos, no obstante si un sistema de agentes implementa métodos que sus agentes invocan y que otros sistemas carecen, dichos agentes no podrán ser ejecutados en esos sistemas.

para que el administrador de cada sistema de agentes pueda otorgar privilegios de forma específica a cada autoridad remota. Así como gozar de una gran versatilidad que otros sistemas de agentes no tienen (AGLETS, D'Agents, etc). Así pues los objetivos de la arquitectura son:

Autentificar los servidores de donde provengan los agentes así como sus autoridades.
Otorgar privilegios de forma individual a los agentes móviles de cada autoridad remota que se quiera reconocer en el sistema.
Lograr especificar los ficheros exactos a los que se permitirá acceder a los agentes de cada autoridad, así como los privilegios de acceso para cada archivo o grupo de archivos.
Poder imponer asignaciones a cada autoridad o sistema remoto para controlar el consumo de recursos en el sistema de los agentes provenientes de sistemas remotos.
Permitir que los agentes puedan definir los privilegios de los ficheros que éstos generen en tiempo de ejecución.
Proteger la transmisión de información de un servidor a otro cuando no se encuentre dentro de una misma región.
Proteger el código del agente mediante firmas digitales y usar la firma de la autoridad de cada servidor para firmar el estado de datos de tal forma que cada servidor que firme el estado de datos adquiera responsabilidad.

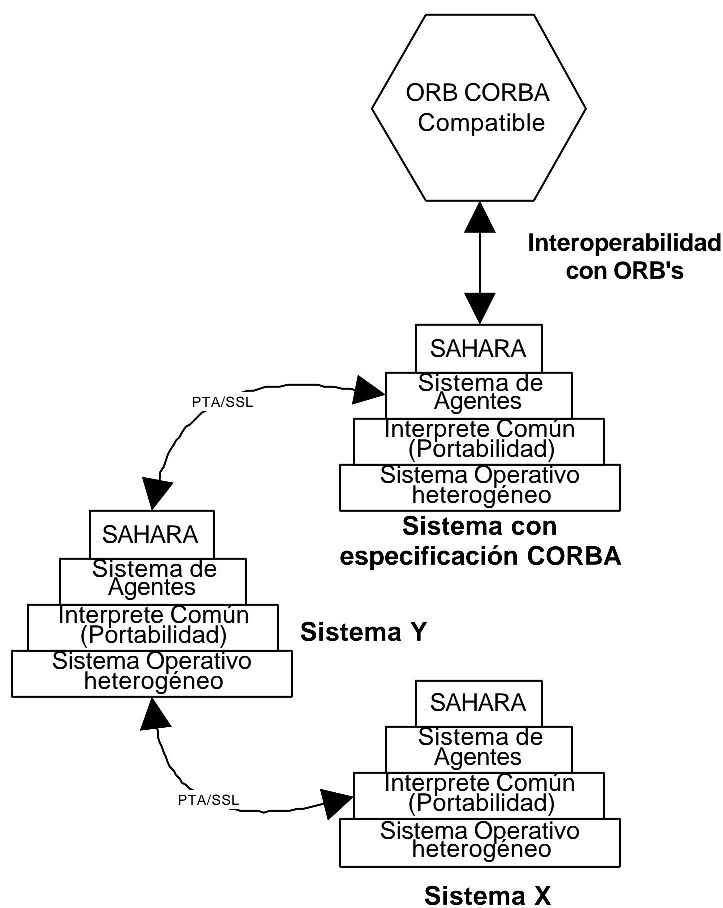


Figura 11.2.1 – Interoperabilidad de sistemas de agentes con SAHARA

Ofrecer mecanismos para la protección del estado de datos de los agentes móviles.

Proveer al servidor de agentes de un gestor inteligente para que modifique las políticas de seguridad en tiempo de ejecución cuando tenga escasez de recursos. Y que las restablezca posteriormente cuando el sistema llegue a ser estable.

11.2.2 Autoridades, usuarios y recursos

Las autoridades que se consideran en la seguridad de SAHARA son los *usuarios de los agentes*, los *fabricantes de los agentes* y las *máquinas servidoras*. Las autoridades serán los encargados de firmar los agentes al ser enviados, son por tanto el elemento fundamental para realizar la autenticación y la asignación de privilegios.

El *usuario del agente* es la persona que inicialmente envía al agente y que es responsable de él, la autorización y la contabilidad de las acciones del agente en un cierto lugar se basan normalmente en esta identidad del usuario.

El *fabricante* de un agente móvil comercial es mejor conocido frecuentemente por otras autoridades que interaccionan con el agente que por su usuario. En realidad los usuarios podrían adjudicar la responsabilidad al fabricante de acciones inesperadas, sobre todo desde del punto de vista legal.

El tercer tipo de autoridad es la *máquina servidora* cuya identidad forma la base en la decisión del agente para migrar a ese destino. La autoridad del servidor es usada para autenticación de servidores.

Cada usuario del sistema tendrá una cuenta, y solo el administrador podrá crear nuevos usuarios/autoridades así como establecer sus asignaciones.

Aunado a las autoridades como sujetos de las acciones están los recursos como los objetos de estas acciones. SAHARA protege los recursos fundamentales del sistema operativo como ficheros, espacio de disco, memoria RAM y el número de conexiones a la red.

Con solo tres clases de autoridades, el modelo de SAHARA es más simple que el propuesto en [FARM96], o el modelo de los aglets [KARJ97] el cual define ocho de ellos. Considero que para efectos prácticos es conveniente proporcionar a los usuarios el menor grado de complejidad para facilitar las tareas.

11.2.3 Uso de lugares para definir contextos de ejecución afines

Un lugar es un contexto dentro de un sistema de agentes en el cual un agente se puede ejecutar. Un sistema de agentes puede contener uno o más lugares y cada lugar puede contener uno o más agentes.

En todos los sistemas de agentes que se implante la arquitectura de SAHARA los lugares aparte de tener la función de agrupación y asociación de agentes afines, jugarán

un papel fundamental para la búsqueda y recopilación de información ya que podrán interactuar intensamente con el resto de los agentes del lugar que tienen objetivos similares. De igual forma los lugares favorecerán el uso compartido de recursos o privilegios, por ejemplo, cada agente dentro de un lugar podrá especificar si desea compartir sus ficheros con los otros agentes del grupo, es decir, del lugar en el que se encuentra, ya que será posible que un agente comparta los ficheros que él va creando especificando inclusive los privilegios para su autoridad, para los agentes del ordenador al que pertenece y para el resto de agentes activos en el sistema. Este enfoque favorece el desarrollo de aplicaciones en paralelo y en conjunto por más de un agente, ya que si un grupo de agentes decide compartir algunos ficheros entre sí mismos podrán lograr un objetivo en conjunto.

De igual manera si un agente del lugar quiere tener exclusividad en su información, también podrá hacerlo al negar estos privilegios al resto de los agentes en su lugar. Obsérvese la figura 11.2.4.1.

11.2.4 Uso de regiones para crear redes confiables

Una región es un conjunto de sistemas de agentes que normalmente tienen la misma autoridad pero que se encuentran en sitios distintos. En la práctica esos grupos de servidores son frecuentemente administrados por una autoridad común tal como una compañía, y puede ser práctico tratar atómicamente a ese grupo de sistemas de agentes desde el punto de vista de seguridad. Por ejemplo omitir las costosas computaciones criptográficas[55] mientras los agentes migran de un servidor a otro dentro de ese grupo.

En SAHARA el concepto de región trata de obtener los beneficios de omitir la criptografía innecesaria sin introducir la complejidad de un nuevo tipo de autoridad. Las regiones no son visibles a los agentes, los cuales simplemente migran entre los servidores como lo hacían anteriormente, es el sistema quien es encargado de definir mediante ficheros de configuración todos los servidores que pertenecen a una región. La interacción es transparente y se muestra en la figura 11.2.4.1.

Cada máquina servidora podrá tener uno a más sistemas de agentes, no obstante en cada máquina existirá el fichero *SAHARA/config/region.cfg* en donde se especificarán todas las direcciones IP o el nombre[56] (incluyendo el dominio) de todas las máquinas servidoras que pertenezcan a la región. Cuando un agente trate de migrar a una máquina dentro de la región, el sistema omitirá algunas validaciones adicionales.

Para que el sistema vaya a buscar dicho fichero bastará con activar una casilla de verificación en la interfaz de configuración para indicarle al servidor de agentes que éste estará incluido en una región. Si la casilla de verificación no se activa el sistema ordenará todo agente que salga o entre sea protegido mediante métodos criptográficos para garantizar el máximo nivel de seguridad, ya que se utilizarían los métodos de

[55] Es grande el tiempo que se pierde en realizar la encriptación de los agentes para ser transmitidos y posteriormente al realizar la autenticación lo que afecta en el rendimiento de las aplicaciones.
[56] Es este caso es necesario que exista un DNS.

autenticación y cumplimiento[57] de privilegios para agentes provenientes de redes inseguras que explicaremos en la siguiente sección.

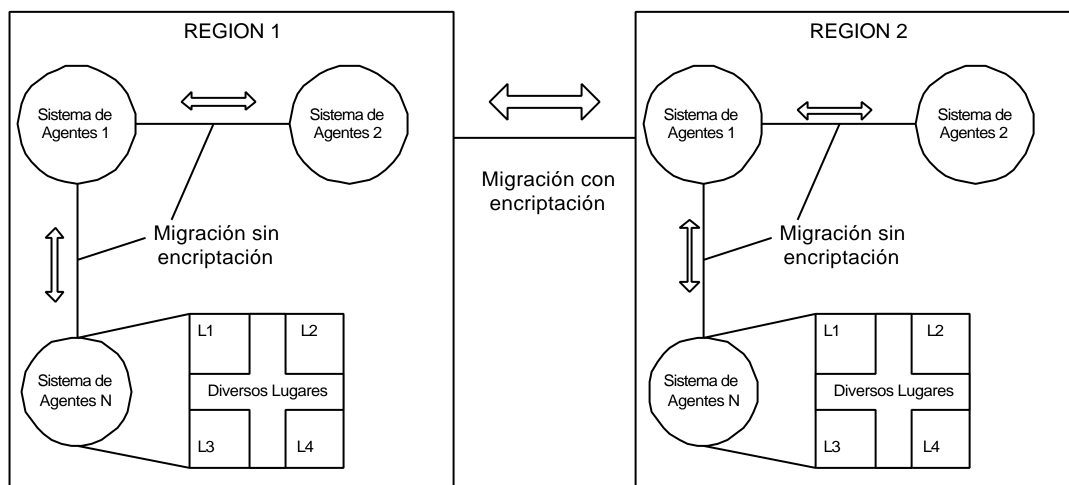


Figura 11.2.4.1 –Migraciones dentro y fuera de una región

11.2.5 Agrupaciones de regiones para crear redes seguras

Quizá la autoridad pueda tener más de una región en ciudades diferentes como el caso de una empresa multinacional. En dicho caso es necesario que las interacciones de los agentes móviles dentro de los diversos sistemas de agentes se realicen con fluidez, para ello es conveniente que más de una región se fusione englobando en un solo grupo a todas las máquinas servidoras, todas gozarán del mismo privilegio, es decir todas las máquinas se considerarán como si pertenecieran a una misma región.

Para lograrlo el sistema tendrá la opción de añadir un fichero llamado *SAHARA/config/ region_group.cfg,* si dicho fichero existe, especificará el nombre de los ficheros en donde se especifican todas las direcciones IP de cada una de las regiones a las que dicho ordenador será enlazado.

Para evitar perdidas de tiempo en la búsqueda de este fichero, el sistema podrá ser configurado para permitir o no el enlace de regiones. Si esta opción se activa, el sistema buscará la existencia de este fichero una vez que un agente pida migrar a una máquina que no se encuentra dentro de *SAHARA/config/region.cfg*, es decir irá a buscar a todas las máquinas que puedan formar parte de la región. Si este fichero no se encuentra, la transferencia del agente se realizará mediante métodos criptográficos, como se realizaría en forma convencional una vez que un agente decide migrar a una máquina de otra autoridad, es decir fuera de la región.

Este esquema es versátil y puede ser inclusive utilizado para unir dos regiones de diferente autoridad cuando el nivel de confianza entre dichas autoridades[58] sea

[57] La frase cumplimiento de privilegios proviene de la palabra inglesa *enforcement* que implica el cumplimiento forzoso de los privilegios asignados a su llegada por el servidor.
[58] Por ejemplo dos compañías del mismo gremio que pertenezcan al mismo dueño.

suficiente para confiar en compartir sus recursos mutuamente. Así mismo el sistema de agentes puede no incluirse dentro de ninguna región o grupo de regiones, para que, como hemos dicho previamente todo agente que salga o entre sea protegido mediante métodos criptográficos para garantizar el máximo nivel de seguridad, ya que se utilizarían los métodos de autenticación y cumplimiento de privilegios para agentes provenientes de redes inseguras que explicaremos en la siguiente sección.

11.2.6 Panorama general del funcionamiento de SAHARA

De acuerdo a lo mencionado anteriormente un sistema de agentes puede sufrir diversos tipos de ataques que pueden resolverse al brindar protección en tres aspectos fundamentales, ya que algunos ataques se traslapan y son resueltos de forma automática al resolver otros. Así pues SAHARA ofrece:

Protección de la máquina o servidor de agentes contra ataques.
Protección de la transmisión de información contra grupos terceros.
Protección de los agentes contra servidores maliciosos.

El esquema global de interacción se detalla en la figura 11.2.6.1 en donde se aprecian los elementos que intervienen para lograr una seguridad integral, los cuales se explican brevemente.

Cada vez que se da de alta un usuario/autoridad en el sistema de agentes se crea un par de llaves que se envían a una entidad certificadora para que corrobore y certifique las llaves, una vez certificadas son almacenas en el repositorio de llaves *Keystore* asociadas con su respectivo usuario/autoridad.

Cada vez que un agente va a migrar se verifica si el servidor de agentes pertenece o no a una región, si pertenece se hacen las comprobaciones necesarias en los ficheros *region.cfg* y *region_group.cfg* y se omiten validaciones entre los servidores. Se obtiene la clave de la autoridad correspondiente del *keystore* y se firma el agente con su asignación adjunta que le fue asociada en su creación del fichero *DBPreferences.* Una vez firmado se envía para que posteriormente pueda ser autentificado.

Cuando un agente remoto es recibido en el sistema, se autentifica mediante su firma digital obteniéndose la llave pública asociada para su validación. Una vez identificada su autoridad y procedencia se observa en los ficheros de políticas de seguridad los permisos que le han sido concedidos y se crea un dominio de protección (salvo que ya existiera uno definido previamente) que define los permisos que tendrá en tiempo de ejecución todos los agentes que procedan de la misma autoridad o servidor y se incluyan en ese dominio.

Una vez identificado los permisos que le corresponden, el agente se asocia a dicho dominio de protección, el cual tiene asociados además de los permisos concedidos un objeto llamado *DBResourcesFree* que indica la cantidad de asignaciones disponibles para dicho dominio de protección.

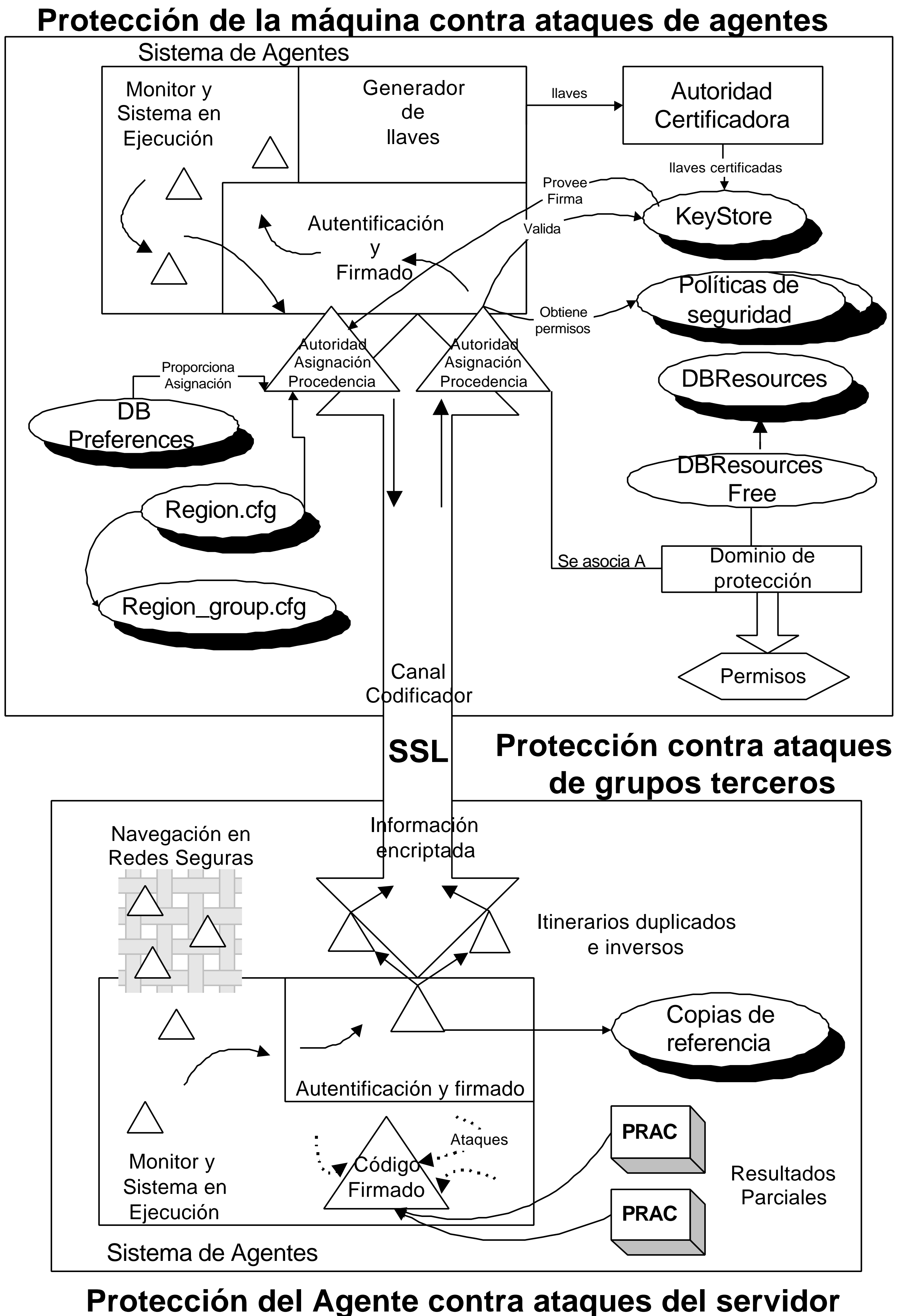


Figura 11.2.6.1 – Esquema global de interacción de SAHARA

Las cantidades de este objeto se actualizan una vez que el agente entra en el dominio. Al crearse el dominio por primera vez el objeto *DBResourcesFree* se crea a partir del archivo *DBResources.*

Para transmitir la información de una forma segura se usa el protocolo SSL (*secure sockets layer*) que ya es un estándar para la transmisión de información en redes mundiales, dicho protocolo establece un canal de comunicación codificado una vez que las máquinas han sido autentificadas previamente.

Durante su ejecución el agente puede dar de alta permisos específicos sobre los ficheros que él vaya creando que facilite el trabajo en grupo y paralelo, en dichos permisos puede determinar a que autoridad(es) y sus permisos de acceso.

Por último para proteger a los agentes contra servidores maliciosos se utiliza como medida fundamental la firma del código de los agentes. Esto nos garantiza que si el código de los agentes es alterado, el siguiente servidor en recibir al agente modificado lo rechazará previniendo con esto un posible mal comportamiento del agente una vez que había sido corrompido. Las firmas digitales en los agentes evitan que ningún servidor de agentes pueda modificarlo.

Como los datos del agente se irán modificando constantemente, éstos no pueden ser firmados por lo que en segunda instancia se definen patrones de itinerarios duplicados que permiten detectar alteraciones en los datos del agente por parte de un servidor malicioso. De igual forma para ciertos itinerarios de búsqueda se crean copias de referencia de los valores de inicialización de los datos de los agentes, a su regreso los datos son verificados con las copias para comprobar que no sufrieron alteraciones.

A continuación explicaré de forma detallada y completa cada uno de los elementos que intervienen en la arquitectura y las tareas que realizan para lograr una seguridad total.

Capítulo XII
SAHARA: PROTECCIÓN DEL SISTEMA DE AGENTES CONTRA ATAQUES

La protección del servidor de agentes contra ataques de agentes móviles o incluso contra ataques de servidores involucra básicamente dos tareas:

Autenticación[59]: es el proceso de deducir que autoridad ha hecho una petición específica. En sistemas distribuidos, la autenticación es complicada por el hecho que una solicitud se puede originar en un servidor remoto y puede atravesar varias máquinas y canales de red que son seguros de diversas maneras y no igualmente confiables.

Autorización y Cumplimiento de Privilegios: el propósito de la autorización es determinar el nivel de privilegios que se le deben proporcionar al agente para realizar su trabajo, esta asignación de privilegios será basándose en la autoridad que pertenezca. Por su parte el cumplimiento de privilegios consiste en vigilar y asegurar que el agente no viole y sobrepase los límites que le han sido asignados por el servidor.

SAHARA como otros sistemas de agentes móviles[60], gestiona la primera tarea mediante el uso de criptografía de llave pública y entornos de ejecución segura, autentificando no sólo a la autoridad del agente sino el servidor del que proviene.

Es importante señalar que el esquema de autorización de SAHARA permite asignar permisos particularmente a cada autoridad remota reconocida en el sistema y señalizar exactamente el tipo de privilegio que se va otorgar sobre el recurso del sistema. La versatilidad para la asignación de privilegios que ofrece SAHARA no la tiene ningún sistema de agentes actualmente. Inclusive sistemas comerciales como Aglets, Concordia y Odissey carecen de esta característica.

[59] Erróneamente muchos autores hispanos usan como sinónimo autentificación, quizá por su similitud con el verbo en castellano autentificar o por su similitud con la palabra inglesa authentication, no obstante la palabra autentificación no existe.
[60] Como D'Agents, ARA y Concordia.

12.1 AUTENTICACIÓN DE AGENTES DENTRO DE UNA REGIÓN

Como he asumido que los sistemas de agentes dentro de una región pertenecen a una misma autoridad y por consiguiente gozan de un nivel de confianza mutuo, no resulta necesario hacer una autenticación particular de dicho sistema sino sólo de las autoridades particulares de los agentes para asignar sus privilegios y considerar la *asignación*[61] que le ha sido impuesta a dicho agente por parte del administrador del sistema de donde proviene.

Diversos usuarios en sistemas de agentes distintos que pertenecen a una misma región podrán tener privilegios diferentes en función de su *asignación* y de los permisos que le sean otorgados a su autoridad particular y a la máquina de donde provienen. No obstante, los agentes pueden ser firmados con una autoridad general que represente a la máquina y que permita otorgar los mismos privilegios a todos los agentes de ese sistema en los ficheros de políticas de seguridad de los sistemas remotos.

Como los privilegios del agente en principio dependerán de los permisos en el fichero de políticas de seguridad y de la asignación que se imponga a cada agente. Se pueden crear entradas con mayor cantidad de recursos en las asignaciones definidas para las máquinas de la región.

La labor de autenticación se apoya directamente en el protocolo SSL para transmisión de datos que se utiliza. De esta forma se elimina la posibilidad de falsificación de autoridades o servidores. No obstante, cada servidor de agentes tendrá siempre la opción de no enlazarse a ninguna región. En el caso de autenticación de servidores dentro de una misma región él protocolo SSL no será usado para agilizar la transmisión.

12.2 AUTENTICACIÓN DE AGENTES PROVENIENTES DE REDES NO CONFIABLES

En este caso no sólo se verifica la máquina de donde proviene el agente sino también la autoridad del agente específico. La autenticación de todos los agentes se hace por medio de su firma digital.

Para lograrlo se hace uso de criptografía de llave pública, esto consiste en usar una formula especial para crear dos llaves que están matemáticamente relacionadas, pero ninguna puede ser deducida de la otra. Una llave (privada) es usada para encriptar un agente en particular para producir un texto encriptado. La otra llave (pública) es usada para desencriptar el agente, sin embargo la llave original no puede ser usada para desencriptar el agente.
La criptografía de llave pública puede ser utilizada en dos esquemas, ambos aportan beneficios y desventajas, describiré brevemente los dos esquemas y posteriormente continuaré describiendo en proceso de autenticación en SAHARA.

[61] Una asignación es un objeto que especifica los privilegios sobre ciertos recursos que tendrá un agente durante su navegación en diversos servidores.

12.2.1 Criptografía de llave pública con distribución de llaves públicas

Bajo este esquema como se señala en [WEBE97] se resuelve el problema de distribución de llaves que limita a la criptografía de llave privada. Un individuo que espera recibir información protegida puede publicar una de sus llaves, generalmente referida como llave pública. Quienquiera que desea enviar información encriptada a esa persona simplemente toma la llave pública y crea un texto cifrado. La información encriptada puede ser trasmitida con seguridad porque sólo la otra llave puede desencriptarla. El receptor mantiene la llave secreta correspondiente, la cual es desconocida para otros, únicamente esta llave puede ser usada para leer la información encriptada por la llave pública correspondiente.

Este esquema proporciona gran versatilidad, y resuelve el problema de distribución. No obstante, presenta una gran desventaja al utilizarlo para autentificar agentes móviles ya que se tiene la certeza de que la información no ha sido alterada pero no sabemos con seguridad quien nos la manda, por consiguiente, no es posible utilizar este esquema para autentificar las autoridades de los agentes móviles.

12.2.2 Criptografía de llave pública mediante firmas digitales

En este esquema dado la información a ser encriptada, la llave secreta comúnmente llamada como llave privada es usada para crear una firma encriptada. La información codificada es trasmitida con la firma y, si el mensaje es alterado, la firma no puede desencriptarlo. Cualquiera quien reciba la información puede obtener la llave pública disponible para asegurar dos cosas:

La información proviene ciertamente del autor supuesto.
La información no ha sido alterada en ninguna forma después de haber sido firmada.

Dicho proceso se puede apreciar en la figura 12.2.2.1. Las firmas digitales como se detalla en [SEBE89] son únicas, infalsificables, fáciles de autentificar, los propietarios no pueden negar su pertenencia y además es barato y fácil generarlas.

Todas estas características las hacen idóneas para utilizarlas con agentes móviles. La arquitectura de SAHARA las utiliza para la autenticación de agentes móviles, no obstante, considera no sólo la autoridad de donde proviene sino también la máquina de procedencia.

12.2.3 Autoridades de certificación

La única limitación en el sistema de llave pública descrito anteriormente es la verificación de que la llave pública verdaderamente pertenece al individuo que se cree. Es concebible que un individuo pueda enviar un agente firmado con una llave secreta, diciendo ser de otro grupo. Este impostor publica la llave pública como si perteneciera a la persona que se pretende falsificar. Entonces se recupera esta llave y la firma se

desencripta creyendo que se ha verificado al susodicho autor y confiando en agentes que han sido escritos por un impostor.

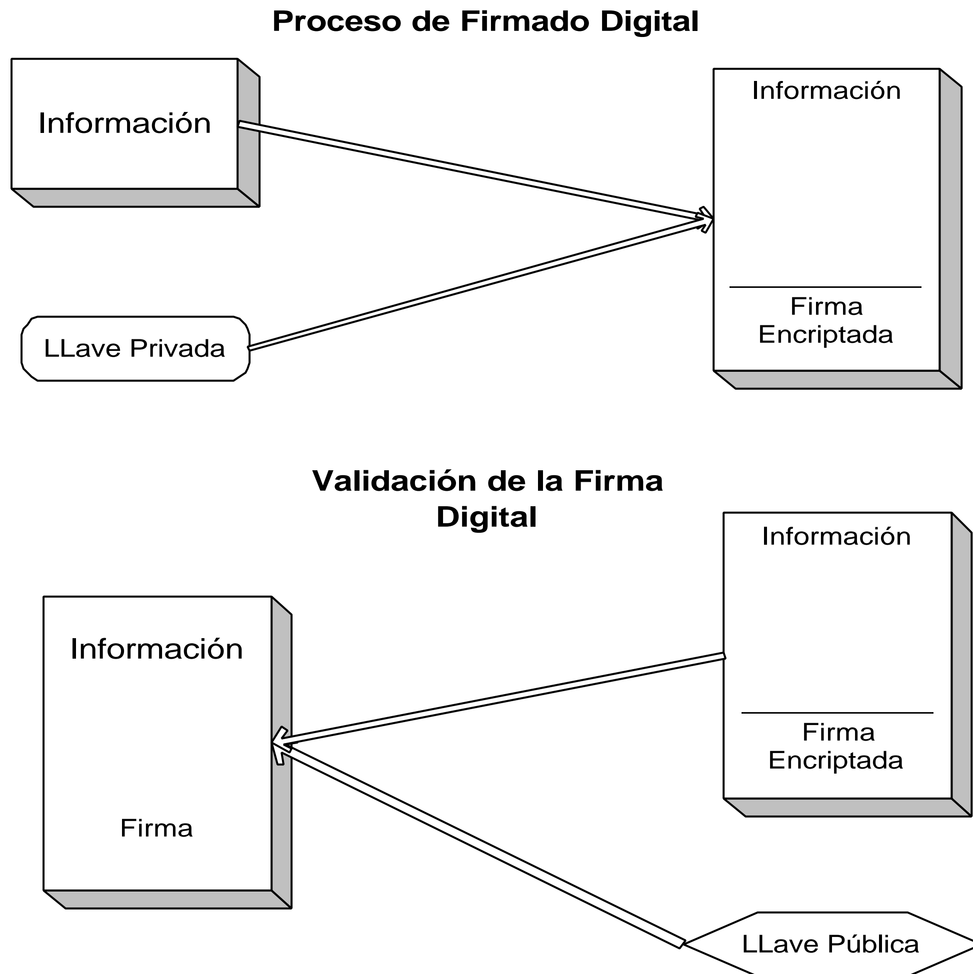


Figura 12.2.2.1 – Firmado y Validación de información mediante firmas digitales

Los sistemas con transmisión segura han recurrido a sistemas conocidos como autoridades de certificación para superar esta limitación. Básicamente, una autoridad de certificación es una organización o compañía que es bien conocida y ampliamente distribuida que asegura que una llave pública es anunciada apropiadamente.

La autoridad de certificación firma la llave de otras entidades que concluyentemente prueba su identidad. Cuando se reciba la llave pública de esta entidad, se puede usar la llave pública de la autoridad de certificación para verificarla. Si se tiene éxito, se sabe que la autoridad de certificación confirma que esta entidad o autoridad es quien dice ser, de esta forma la autoridad de certificación certifica a dicha entidad.

La arquitectura de SAHARA es flexible para realizar la certificación de firmas, se puede recurrir a una entidad certificadora como *Netscape* o bien a una entidad certificadora que proporcionará la propia arquitectura de seguridad, que estará centralizada en una máquina especifica a donde todos los sistemas de agentes podrán recurrir.

Dicha entidad certificadora será confiable para todos los sistemas de agentes que pertenezcan a una región o a un grupo de regiones. Dependiendo del alcance de aplicación se deberá hacer uso de una autoridad certificadora oficial o bien de una autoridad certificadora para el grupo de regiones.

Para su localización habrá un fichero *SAHARA/config/CA.cfg*, que contendrá el nombre o la dirección de la autoridad certificadora que se usará dentro de la región para certificar las firmas en caso de que no se use una autoridad certificadora oficial.

12.2.4 Almacenamiento de las llaves y certificados de cada autoridad

Como explicaré más delante el uso de llaves y certificados es vital para el proceso de autenticación, por ello SAHARA define a una base de datos llamada *keystore* para que pueda ser usado para gestionar un repositorio de llaves y certificados[62].

El formato de dicha base de datos varia de acuerdo a la proveedor de seguridad que se utilice, SAHARA utilizará el formato JKS que es el utilizado por defecto en la versión del JDK 1.2 sobre el cual será implementado. Este formato protege cada llave privada con una contraseña individual y también protege la integridad de toda la base de datos con otra contraseña.

La base de datos *keystore* gestiona dos tipos de entradas:

Entrada de llave: este tipo de entrada mantiene información de una llave que es muy sensible criptograficamente, por lo cual es almacenada en un formato protegido para prevenir acceso no autorizado. Típicamente, una llave almacenada en este tipo de entrada es una llave secreta, o una llave privada acompañada por el certificado de autenticación correspondiente a la llave pública. Las llaves privadas y los certificados son usados por una entidad dada para una auto-autenticación usando firmas digitales
Entrada de certificado: este tipo de entrada contiene un certificado individual de llave pública perteneciente a otro grupo. Esta entrada es llamada certificado confiable porque el propietario de la base de datos *keystore* confía en que la llave pública en el certificado en verdad pertenece a la identidad identificada por el sujeto(propietario) del certificado.

Cada entrada en el *keystore* es identificado por una cadena “alias”. En el caso de las llaves privadas y sus certificados asociados, estas cadenas distinguen a los diferentes usuarios o autoridades que podrán firmar agentes dentro del sistema para que éstos a su vez naveguen en su nombre.

En el caso de las llaves públicas representan a los usuarios o autoridades remotas que podrán ser autentificadas en el sistema local.

Dentro del proceso de autenticación de agentes el sistema accederá a la base de datos *keystore* para acceder a las llaves necesarias para firmar a los agentes que van a migrar y autentificar a los agentes que van llegando.

[62] Un certificado es una declaración firmada digitalmente de una entidad, diciendo que la llave pública de alguna autoridad tiene un valor particular.

12.2.5 Firmado de agentes móviles

Como mencioné anteriormente una vez que un agente o cualquier información es firmada, ésta ya no puede ser modificada ya que al realizar la validación el agente será rechazado si su contenido es alterado.

Debido a esto se deben considerar las partes que conforman a un agente móvil, como lo es su estado de código, su estado de datos y el estado de ejecución[63]. Solo la parte inmutable del agente podrá ser firmada, dicha parte no podrá ser alterada y si algún servidor malicioso la modifica, el siguiente servidor que reciba al agente podrá darse cuenta que ha sido alterado al realizar la validación correspondiente mediante la llave pública de la autoridad que lo firma.

De esta forma SAHARA sólo puede firmar la parte inmutable del agente que incluye el código y todos aquellos datos que son constantes y que no van a ser modificados durante el itinerario del agente. Una vez firmada la parte inmutable se garantiza su integridad contra cualquier ataque por servidores maliciosos o grupos terceros. Y el agente podrá navegar con toda seguridad por un grupo indeterminado de servidores sin ser alterado.

El servidor realizará la separación necesaria de información al ser llamado el método *go()*[64] antes de hacer la serialización del agente para que los datos puedan ser firmados apropiadamente.

Aunque el estado de datos o parte mutable del agente será protegido mediante otras técnicas que se presentarán más delante en este trabajo, éste puede ser firmado con la llave correspondiente a la autoridad de cada servidor para adjudicarle responsabilidad, es decir a la hora de migrar el servidor firmara el estado de datos del agente garantizando con ello que el no modifico o altero los datos del agente.

Para que una autoridad pueda firmar agentes, será necesario que este dada de alta en la base de datos *keystore*. Esto el sistema lo hará automáticamente cada vez que un nuevo usuario sea dado de alta en el sistema, para ello realizará lo siguiente:
Creará un par de llaves para dicha autoridad.
Enviará a certificar las llaves a la autoridad certificadora definida en el sistema.
Añadirá la llave y el certificado al *keystore* con su respectivo 'alias' que identificará a la autoridad específica.

De esta forma cada vez que un usuario quiera enviar un agente a navegar, el sistema usará su firma respectiva para firmarlo y poderlo enviar de una forma segura.

[63] El cual en los sistemas de agentes basados en Java es imposible de transportar debido a que es imposible hacer accesos a su pila.
[64] Que permite que el agente migre

12.2.6 Proceso de autenticación en SAHARA

El esquema de autenticación de SAHARA se basa fundamentalmente en la firma digital adjunta a los ficheros que forman el agente y que se usa para verificar el código firmado que representa al agente y que se originó en el sistema de agentes remoto. SAHARA busca en el *keystore* el certificado de la autoridad correspondiente para realizar la autenticación y verificación de la firma.

SAHARA además considera para autentificar al agente el **CodeSource (Origen del código)**[65] que es un objeto que encapsula la autoridad y dirección (si se añade) de donde proviene el agente. Cada entrada en el fichero de políticas de seguridad del sistema consiste de un *codesource* y sus permisos. Cada entrada incluye la lista de las autoridades que firman y la URL de donde se podrían cargar clases requeridas por el agente, por ejemplo:

```
Grant [SignedBy “Nombres alias de quienes firman”] [ ,CodeBase “URL”] {

        PERMISOS
}
```

Si en el campo *signedby* se añade a la autoridad la dirección IP donde se creo el agente, éstos serán autentificados no sólo en función de la autoridad a la que pertenezcan sino del sitio de donde provengan. La URL del *Codebase* será raramente usada para autoridades remotas, sólo por aquellos agentes que ejecuten grandes aplicaciones que necesiten código adicional que no hayan traído con ellos mismos (por su tamaño) y requieran descargarlo de la red.

SAHARA se encarga de crear el *codesource* correspondiente después de consultar el *keystore* para determinar los certificados correspondientes de los firmantes especificados.

De esta forma el cargador de clases podrá hacer uso del *codesource* para validar los agentes provenientes de sitios remotos y autentificar a sus respectivas autoridades. Si alguna de las autoridades especificadas en el fichero de seguridad no existe en el *keystore*, el código será considerado como inseguro. Si la validación de la firma de alguna autoridad falla dicho agente no podrá entrar al sistema ya que ha sido alterado.

De esta forma SAHARA asegura que ningún agente cuya autoridad no haya sido autentificada podrá gozar de ningún permiso. De hecho la arquitectura de seguridad puede ser configurada para rechace a todos los agentes no autentificados o los trate de una forma muy especial en la que restringe completamente su funcionamiento.

[65] Que se refiere a la autoridad a que pertenece y al lugar de donde proviene el código que permitirá reiniciar al agente.

12.2.7 ¿Qué se logra mediante la autenticación?

El esquema de autenticación de SAHARA permite asegurar que ningún agente podrá navegar en un sistema en el cual no haya sido autorizado para hacerlo. Aunque la arquitectura es lo suficientemente versátil para permitir que un agente pueda ser trasmitido sin el uso de criptografía dentro de una región, la transferencia se hace mediante el protocolo SSL que internamente realiza una autenticación de máquinas y posteriormente encripta la información para ser transmitida.

Más aún SAHARA puede configurarse para que todo servidor de agentes sea independiente y no exista ninguna región de por medio, con ello se lleva un doble mecanismo de seguridad, el protocolo de transferencia y las firmas digitales. Debido a este doble mecanismo de seguridad se puede asegurar que SAHARA no es vulnerable a ataques de agentes en contra de la máquina o servidor y a ataques de máquinas remotas contra el servidor de agentes, ya que antes de que se establezca cualquier relación entre ellas estas tienen que ser previamente autentificados.

12.3 AUTORIZACIÓN

Una vez que ha sido deducida la autoridad del agente es necesaria la **autorización**, es decir, asignar los privilegios apropiados basándose en dicha autoridad. Para los agentes que pertenecen a la misma región se hará con relación a la máquina de donde provienen sin importar que usuario los ha firmado. A los agentes que provienen de regiones diferentes les serán asignados los privilegios dependiendo del sitio del que provienen y de la autoridad o autoridades que los hayan firmado.

Los privilegios que se otorgan tienen como finalidad proteger el uso excesivo de los recursos de la máquina servidora por parte de los agentes como lo son el sistema de ficheros, RAM y conexiones a la red, de igual forma pretenden evitar el posible robo de recursos entre los agentes que se encuentran activos en el sistema.

Entre los recursos que se les otorgarán a los agentes podemos clasificarlos en función de la política de **cumplimiento de privilegios** que se utilizará para restringirlos, así pues podemos citar:

Privilegios controlados directamente por el sistema de agentes.
Privilegios controlados desde el fichero de políticas de seguridad del sistema de agentes.

Los privilegios normalmente se traslapan, por ejemplo, el fichero de políticas de seguridad determina las rutas del sistema de archivos a las que un agente móvil puede acceder mientras el sistema de agentes controla la cantidad de kilobytes que el agente móvil puede escribir.

SAHARA a diferencia de ARA[66], contabiliza por medio de las asignaciones el número de recursos consumidos por una autoridad o servidor para poder limitar la cantidad de

[66] Que sólo permite el uso de asignaciones por agente y por tanto no determina las cantidades consumidas por una autoridad o servidor específico

recursos consumidos por los mismos. El resto de los sistemas de agentes en el mercado no hace uso de asignaciones.

Así pues describiré como se realiza la gestión de cada tipo de privilegios, no sin antes describir las propiedades que posee un agente y todo el tipo de privilegios que un agente móvil puede tener.

12.3.1 Uso de asignaciones para restringir el acceso a recursos solicitados por el usuario y controlados por el sistema

Dentro de la estructura general del agente móvil podemos señalar algunos elementos básicos:

Atributos privados:

```
NameMobileAgent  Name          // nombre del agente
Itinerary  Recorrido           // itinerario
```

En donde la clase *NameMobileAgent* contiene los siguientes atributos:

```
NameAgentSystem  AgentSystem   // datos del sistema de agentes donde fue creado
String AgentClass              // nombre de la clase a partir de la cual se creó
String AgentAuthority          // propietario del agente
Date CreationDate              // fecha y hora de creación del agente
```

A su vez la clase *NameAgentSystem* define los atributos de identificación de un sistema, por lo que se incluyen los atributos:

```
String Authority               // propietario del sistema de agentes
String AgentSystemId           // identificador único del sistema de agentes
String IP                      // dirección IP del sistema de agentes
```

Cada agente móvil contará con un objeto *Nombre* que define sus características principales y con un itinerario específico. En el momento de su creación le serán asociados un identificador[67] único y un objeto llamado **asignación** de tipo *ResourcesAvailable* que indica las restricciones que tendrá el agente una vez que migre a un servidor remoto. Entre ellas se podrá restringir el número de agentes hijos que podrá crear y el número de KB que el agente podrá usar de espacio en disco en el sistema remoto. Los campos del objeto asignación son los siguientes:

```
Int Disco              // Cantidad en kilobytes que puede usar
Int Hijos              // Número de hijos que puede crear el agente
Int Memoria            // Cantidad en kilobytes que puede usar
Boolean Mensajes       // Capacidad o no de enviar mensajes
Boolean Duplicar       // Capacidad o no de duplicarse
Boolean Share          // Capacidad o no de compartir ficheros
```

[67] El que servirá para darlo de alta y localizarlo en la tabla de referencias de agentes. Véase capítulo XIII.

Para los dos primeros casos el valor de cero implicaría que no gozan del privilegio, en el caso del espacio en disco, aunque la autoridad gozará de permisos de escritura en el fichero de políticas de seguridad del sistema, esta opción tendrá prioridad por lo que no podría realizar operaciones de escritura. La memoria tendrá que tener un valor mayor a cero, y los dos últimos parámetros serán utilizados para determinar si el agente podrá o no duplicarse y enviar mensajes.

El administrador de un servidor de agentes tendrá la facultad de asociar una *asignación* distinta a cada usuario/autoridad del sistema de tal forma que permita que los agentes de cada usuario puedan gozar de más o menos privilegios. Las asignaciones establecidas para los usuarios locales estarán definidas en SAHARA/Security/ **DBPreferences.**

Para evitar que un administrador otorgue *asignaciones* muy grandes a sus usuarios, cada sistema de agentes contará con una base de datos llamada SAHARA/Security/ **DBResources** en donde se especifican las cantidades máximas de memoria, espacio en disco y agentes hijos permitidos para cada sistema o autoridad remota, cada vez que un agente provenga de sistema de agentes específico o bien cuando un agente se duplique, se actualizarán los valores correspondientes a dicho sistema o autoridad, de tal forma que los agentes provenientes de un mismo sistema de agentes o de una misma autoridad podrán navegar sin ningún problema dentro del sistema remoto mientras no agoten los recursos que le han sido impuestos en el servidor remoto.

Este esquema obliga a que un administrador asigne cuidadosamente las *asignaciones* a sus usuarios de tal forma que trate de ahorrar en la medida de lo posible recursos que estarán controlados en un servidor remoto. Es decir, solo otorgará asignaciones con muchos recursos a aquellos usuarios que en verdad lo requieran.

Por último los agentes provenientes de servidores desconocidos también serán contabilizados de igual manera, salvo que en vez de que exista un registro en **DBResources** para dicho sistema, se usará un valor por defecto para servidores no autentificados, el cual podría ser un poco menor o muy restrictivo, pero suficiente para que los agentes puedan realizar sus tareas.

12.3.2 Autorización de las asignaciones

El agente viajará con su *asignación* lo que permitirá al agente, controlar algunos de sus recursos en servidores remotos y al sistema de agentes contabilizar los recursos de los agentes móviles que provienen de un sistema específico o de una autoridad, y con ello, poder restringir dichas cantidades a un sistema de agentes o a un propietario de ellos.

Un sistema de agentes autorizará las cantidades solicitadas por la asignación del agente siempre y cuando dichas cantidades no excedan los límites máximos fijados para el sistema de agentes del que proviene o para la autoridad a la que pertenece. Dichos límites se irán contabilizando.
Este esquema es una forma eficaz de proteger a una red de servidores de agentes móviles, ya que si hubiera un servidor malicioso que quisiera agotar los recursos del resto de los servidores no podría sobrepasar más allá de los límites que le sean impuestos en cada servidor de la red.

12.3.3 Tipos de permisos para los recursos del sistema

Antes de poder definir un mecanismo autorización y cumplimiento de privilegios para los recursos del sistema es necesario definir cuales son los privilegios que podrán ser otorgados a todo agente móvil desde los ficheros de políticas de seguridad del sistema, para ello se han considerado los recursos fundamentales que suelen ser indispensables proteger en un sistema:

Sistema de ficheros: el formato para esta clase de permisos puede ser especificado de varias maneras que a continuación se presentan, en donde el nombre de los directorios y de los archivos no puede contener espacios en blanco:

```
archivo
directorio (lo mismo que directorio/)
directorio/archivo
directorio/* (todos los archivos en este directorio)
* (todos los ficheros en el directorio actual)
directorio/- (todos los ficheros en el sistema de archivos en este directorio)
- (todos los ficheros en el sistema de archivos del directorio actual)
"<<ALL FILES>>" (todos los ficheros en el sistema de archivos)
```

Observe que <<ALL FILES>> es una cadena especial que denota todos los ficheros en el sistema. En un sistema Unix, esto incluye todos los ficheros debajo del directorio raíz. En un sistema MS-DOS, esto incluye todos los archivos en todos los discos (incluyendo disketes).

Las acciones a ejecutar son: leer (read), escribir (write), borrar (delete) y ejecutar (execute), a continuación se muestran ejemplos válidos para crear permisos.

```
FilePermission p = new FilePermission("myfile", "read,write");
FilePermission p = new FilePermission("/home/arturop/", "read");
FilePermission p = new FilePermission("/tmp/mytmp", "read,delete");
FilePermission p = new FilePermission("/bin/*", "execute");
FilePermission p = new FilePermission("*", "read");
FilePermission p = new FilePermission("/-", "read,execute");
FilePermission p = new FilePermission("-", "read,execute");
FilePermission p = new FilePermission("<<ALL FILES>>", "read");
```

Es importante señalar que el sistema considera todas las implicaciones que existan respecto a los anidamientos de directorios. Por ejemplo, *FilePermission("/-", "read,execute")* implica *FilePermission("/home/arturop/public_html/index.html", "read")*, y *FilePermission("bin/*", "execute")* implica *FilePermission ("bin/emacs19.31", "execute")*.

Es necesario que las cadenas sean dadas en un formato dependiente de la plataforma hasta que exista un lenguaje de descripción de ficheros universal. Por ejemplo para representar el acceso de lectura de una fichero llamado “foo” en el directorio “temp” en la unidad C de un sistema Windows se debe de usar:

```
FilePermission p = new FilePermission("c:\\temp\\foo", "read");
```

Debe observarse que un nombre que sólo especifique un directorio, con una acción "read" como en:

```
FilePermission p = new FilePermission("/home/arturop/", "read");
```

Sólo esta dando el permiso para listar los ficheros en ese directorio, y no de leer alguno de ellos. Para permitir el acceso de lectura de los ficheros, se debe especificar explícitamente un nombre de archivo, un "*" o un "-"

```
FilePermission p = new FilePermission("/home/arturop/myfile", "read");
FilePermission p = new FilePermission("/home/arturop/*", "read");
FilePermission p = new FilePermission("/home/arturop/-", "read");
```

Por último, debido a que los agente móviles de una misma autoridad normalmente realizarán tareas compartidas el sistema por defecto define que todo agente siempre tiene permiso de forma automática de leer ficheros de su mismo *CodeSource* (y subdirectorios de ese *CodeSource)*, éste no necesita permiso explícito para hacerlo.

Acceso a la red: el formato para este permiso puede ser especificado como "**hostname:port_range**", donde el nombre del servidor (hostname) puede ser dado de las siguientes maneras:

```
hostname (un servidor individual)
IP address (un servidor individual)
localhost (la máquina local)
"" (equivalente a "localhost")
hostname.domain (un servidor individual dentro del dominio)
hostname.subdomain.domain
*.domain (todos los servidores en el dominio)
*.subdomain.domain
* (todos los servidores)
```

Esto es, el servidor es expresado con un nombre DNS, como una dirección IP o como un *localhost*. El rango de puertos (*port_name*) puede ser dado como sigue:

```
N (un puerto simple)
N- (todos los puertos cuyo número sea a partir de N y hacia arriba)
-N (todos los puertos cuyo número sea a partir de N y hacia abajo)
N1-N2 (todos los puertos entre N1 y N2, los límites inclusive)
```

En donde N, N1 y N2 son números enteros no negativos de 0 a 65535.

Las acciones sobre *sockets* son aceptar (*accept)*, conectar (*connect)*, escuchar (*listen)* y resolver (*resolve, que es un DNS).* Note que la acción "resolver" se requiere implícitamente al "aceptar", "conectar" y "escuchar". La acción de "escuchar" se aplica sobre puertos del servidor local, mientras que "aceptar" puede usarse en puertos de un servidor local o remoto. A continuación se muestran unos ejemplos válidos de estos permisos:

```
SocketPermission p = new SocketPermission("java.uniovi.es","accept");
p = new SocketPermission("204.160.241.99","accept");
p = new SocketPermission("*.com","connect");
p = new SocketPermission("*.uniovi.es:80","accept");
p = new SocketPermission("*.uniovi.es:-1023","accept");
p = new SocketPermission("*.uniovi.es:1024-","connect");
p = new SocketPermission("java.uniovi.es:8000-9000","connect,accept");
p = new SocketPermission("localhost:1024-","accept,connect,listen");
```

Permisos en tiempo de ejecución y ventanas: el formato para estos permisos está representados por una cadena válida y no hay acciones asociadas. Por ejemplo, el permiso de tiempo de ejecución "exitVM" denota el permiso de abortar la máquina virtual de java. Los valores válidos son:

```
createClassLoader
getClassLoader
setContextClassLoader
exitVM
setFactory
setIO
modifyThread
stopThread
modifyThreadGroup
getProtectionDomain
readFileDescriptor
writeFileDescriptor
queuePrintJob
showWindowWithoutWarningBanner
accessEventQueue
```

Es importante mencionar que estos permisos muy rara vez serán otorgados a un agente, no obstante se brinda la oportunidad de poder hacerlo.

Permisos de Seguridad: los permisos de seguridad controlan el acceso a los objetos relacionados con seguridad, tales como objetos de política de seguridad(policy), proveedor (provider), firmante (signer) e identidad (identity). Se pueden solicitar los siguientes privilegios:

```
getPolicy
setPolicy
getProperty.{llave}
setProperty.{ llave }
setIdentityPublicKey
setIdentityInfo
printIdentity
addIdentityCertificate
removeIdentityCertificate
getSignerPrivateKey
setSignerKeyPair
```

Todos los permisos (java.security.AllPermission): existe un permiso que implica todos los permisos, este es introducido para si se desea otorgar a los agentes móviles creados por el administrador todos los privilegios necesarios para desempeñar cualquier tarea. Debe tenerse un especial cuidado en otorgar este tipo de privilegio.

12.3.4 Aclaraciones sobre la Implicación de permisos

Los permisos son con frecuencia comparados unos con otros, y en ocasiones la implicación de permisos resulta obvia, por ejemplo el permiso *java.io.FilePermission ("/tmp/*", "read")* implica el permiso *java.io.FilePermission("/tmp/a.txt", "read")* pero no implica ningún permiso de red *java.net.NetPermission*.

Sin embargo, existe otro nivel de implicación que no puede ser obvio inmediatamente para muchos usuarios. Supóngase por ejemplo que a un agente le ha sido concedido el permiso de escribir completamente en el sistema de ficheros. Este por supuesto permite que el agente remplace el sistema binario, incluyendo el entorno de la máquina virtual de Java en tiempo de ejecución. Esto significa que efectivamente al agente le han sido concedidos todos los privilegios.

Es importante observar que otros permisos que son muy peligrosos de otorgar son aquellos que permiten el establecer las propiedades del sistema, definición de paquetes en tiempo de ejecución, para cargar librerías de código nativo y por supuesto el permiso "*allpermission*".

12.3.5 Formato del fichero que define la política de seguridad y la asignación de privilegios

Para establecer la política de autorización que un sistema de agentes va a definir se especifican uno o más ficheros de configuración. Los ficheros de configuración de seguridad especifican que permisos son concedidos a agentes de diferentes *codesources.*

Estos ficheros representan el mecanismo básico de autorización de SAHARA, ya que en ellos se pueden especificar en detalle los privilegios que les serán concedidos a los agentes de una misma autoridad y a los agentes provenientes de un mismo servidor.

Un fichero de configuración de la política de seguridad[68] llamado *SecurityPolicy* esencialmente contiene una lista de entradas. El archivo puede contener un *keystore* y cero o más entradas *grant*[69].
Un *keystore* es una base de datos que contiene las llaves públicas con sus correspondientes certificados que autentifican las llaves públicas correspondientes. El *keystore* especificado en el fichero de configuración es usado para buscar las llaves públicas de los firmantes en las entradas *grant* del fichero. Un *keystore* debe aparecer en

[68] También llamado de asignación de privilegios.
[69] Grant es la palabra reservada que se utiliza para iniciar una nueva entrada de definición de privilegios para una nueva autoridad o servidor.

el fichero si al menos una entrada *grant* especifica que los agentes de dicho *codesource* vienen firmados.

Sólo puede haber una entrada *keystore* en el fichero de asignación de privilegios, si hubiera las de una sólo la primera es considerada y las demás ignoradas. La sintaxis para esta entrada es:

```
keystore "some_keystore_url", "keystore_type";
```

Donde "*some_keystore_url*" especifica la localización URL del *keystore* y *"keystore_type"* especifica su tipo.

La URL es relativa a la localización del fichero de políticas de seguridad, para los sistemas de agentes basados en Java la localización del fichero de seguridad será definido en el fichero de propiedades de seguridad.

El tipo del *keystore* define el formato de los datos y del almacenamiento de la información, así como los algoritmos usados para proteger las llaves privadas y la integridad de la base de datos. Puede especificarse cualquier formato estándar apropiado al lenguaje de la implementación, en el prototipo que se implementará se usará JKS que es compatible con Java.

Cada entrada *grant* en el fichero de políticas de seguridad consiste de un *codesource* y sus permisos. La palabra *grant* es reservada e indica el comienzo de una nueva entrada. Dentro de cada entrada la palabra *permission* es otra palabra reservada que señala el comienzo de un nuevo permiso. El formato es el siguiente:

```
grant [SignedBy "nombres de los firmantes"] [, CodeBase "URL"] {
   permission permission_class_name [ "target_name" ]
            [, "action"] [, SignedBy "nombres de los firmantes"];
   permission ...
};
```

El orden de los campos de la dirección de procedencia (*codebase)* y el firmante (*signedby)* no importa. Y los espacios en blanco son permitidos antes y después de las comas. El campo del *codebase* es opcional, y será omitido en la mayor parte de los casos ya que los agentes normalmente serán enviados con las clases que requieran para volver a ser creados en los servidores remotos. Quizá sólo lo usarían agentes que eventualmente pudieran requerir una librería adicional de su lugar de origen.

El primer campo para el firmante es una cadena 'alias' que se hace corresponder con un conjunto de llaves que están asociadas con esos firmantes. Este campo puede contener múltiples firmantes separados por comas, la relación será entre ellos será '**y** " no '**o",** es decir, significará que los agentes fueron firmados por todos y no por uno de ellos. El campo es opcional, si es omitido significa "cualquier firmante" o que no importa si el agente viene firmado o no.
Una gramática informal en BNF del formato del fichero de políticas de seguridad se muestra a continuación, en donde, las palabras no comenzadas en mayúsculas son terminales.

```
PolicyFile -> PolicyEntry | PolicyEntry; PolicyFile
PolicyEntry -> grant {PermissionEntry}; |
        grant SignerEntry {PermissionEntry} |
        grant CodebaseEntry {PermissionEntry} |
        grant SignerEntry, CodebaseEntry {PermissionEntry} |
        grant CodebaseEntry, SignerEntry {PermissionEntry} |
        keystore "url"
SignerEntry -> signedby (una lista de cadenas separadas por comas)
CodebaseEntry -> codebase (una cadena representando una URL)
PermissionEntry -> OnePermission | OnePermission PermissionEntry
OnePermission -> permission permission_class_name
         [ "target_name" ] [, "action_list"]
         [, SignerEntry];
```

Así pues, la siguiente entrada concede dos permisos a los agentes firmado por DptoInformática (Departamento de Informática) y Rolando:

```
grant signedBy "DptoInformática, Rolando" {
  permission FilePermission "/tmp/*", "read";
  permission PropertyPermission "exitVM";
};
```

Aquí se conceden los permisos a los agentes que son firmados por Microsoft y que pueden requerir clases adicionales que obtendrán a través de la red de la dirección www.uniovi.es:

```
grant codeBase "http://www.uniovi.es/*", signedBy "Microsoft" {
    permission FilePermission "/tmp/*", "read";
    permission SocketPermission "*", "connect";
}
```

Los nombres de los firmantes podrían incluir su dominio o la dirección IP de donde provienen para que la asignación de permisos fuera en función de la autoridad y de la máquina de procedencia. De igual forma sirven para evitar la duplicidad en los ‘alias’ del *keystore* y para mostrar fácilmente su procedencia.

12.3.6 Expansión de propiedades en los ficheros de políticas de seguridad

La expansión de propiedades es posible en el fichero de políticas de seguridad. La expansión de propiedades es similar a la expansión de variables en un shell. Esto es, cuando una cadena como:

“${alguna.propiedad}”

Aparezca en el fichero de políticas de seguridad, será expandido al valor de la propiedad del sistema. Por ejemplo:

```
permission FilePermission "${user.home}", "read";
```

Expandirá "${user.home}" para usar el valor de la propiedad "user.home" del sistema. Si el valor de la propiedad es "/home/candi", entonces la línea de arriba será igual a:

```
permission FilePermission "/home/candi", "read";
```

Obsérvese que las propiedades anidadas no son permitidas por lo que algo como “${user.${foo}}” no funcionará.

Con la finalidad de ayudar a la independencia de la plataforma de los ficheros se puede usar la notación especial de “${/}”, la cual es una abreviación de “${file.separator}” (separador de las rutas de ficheros).

12.3.7 Autorización de privilegios para los agentes locales

Las asignaciones normalmente proporcionan ciertos privilegios que son controlados directamente por el sistema, por su parte los ficheros de políticas de seguridad permiten gestionar de forma más específica el acceso a los recursos del sistema y definir si la autorización de los privilegios será en función de una autoridad, de un servidor de agentes o de ambos.

Obviamente los agentes locales al ser creados no son firmados por su autoridad sino hasta que ellos migran a un nuevo servidor. Al ejecutarse el agente se le otorgan privilegios en función del directorio (*codebase*) dentro del sistema de ficheros de donde fue intanciada su clase. Es decir, el código ejecutado desde una carpeta determinada tiene los privilegios que le han sido otorgados a ese mismo directorio.

El administrador del sistema puede decidir en otorgar los mismos privilegios a todos los usuarios locales, o bien, siguiendo la filosofía de SAHARA (que pretende otorgar privilegios particulares a todos los usuarios) crear un directorio particular de lanzamiento de agentes para cada usuario definido en el sistema dentro de la carpeta SAHARA/Users.

Al tener cada usuario una carpeta individual se añade una entrada *grant* en el fichero de políticas de seguridad en donde se especifican los permisos que se le concederán a los agentes de dicho usuario en el sistema local. Por ejemplo:

```
grant codebase “file:\\SAHARA\Users\arturop” {
  permission FilePermission "/aplicaciones/*", "read, write";
};
```

La entrada anterior concede permisos de lectura/escritura sobre los ficheros del directorio de \aplicaciones a todos los agentes que sean creados a partir de las clases que se encuentran en el directorio \SAHARA\Users\arturop que evidentemente pertenecen al usuario arturop.

Es importante señalar que en las entradas *grant* del fichero de políticas de seguridad se usará la palabra *codebase* para asignar privilegios a los usuarios locales, mientras que se usará *signedBy* para asignar privilegios a las autoridades remotas.

12.3.8 Autorización de privilegios para los agentes que provienen de una misma región

Para los agentes que provienen de una misma región, la autorización de privilegios especificada en el fichero de políticas de seguridad podrá estar en función de su procedencia. Es decir, se verificará que procedan de una dirección específica, por ejemplo:

```
grant signedBy “156.35.31.66" {
  permission FilePermission "/tmp/*", "read";
};
```

De esta forma, los privilegios serán otorgados de forma particular a cada máquina que pertenezca a la región, sin importar si vienen firmados o no por una autoridad particular. No obstante podrá especificarse una autoridad específica en el campo *signedby* para que una autoridad dentro de una región tenga permisos especiales.

12.3.9 Autorización de privilegios para los agentes que provienen de redes no confiables

Para los agentes que provienen de cualquier sitio exterior a la región, la asignación de privilegios estará basada en primer instancia en la autoridad que les firma, por lo que todos los agentes que dicha autoridad firme, gozarán de los mismos privilegios. Por ejemplo:

```
grant signedBy "Grupo_Oviedo3 " {
   permission FilePermission "/tmp/*", "write";
   permission SocketPermission "*", "accept";
};
```

Una vez que un agente comienza a navegar su código va protegido, sin embargo sus datos (aunque como veremos posteriormente se podrá verificar su integridad) podrían ser potencialmente alterados por un servidor malicioso. Para gran parte de la comunidad de agentes móviles que indaga en el tema de seguridad, la autorización de privilegios debería estar en función no sólo de la autoridad que lo firma sino del sitio o sitios de donde provenga, como se señala en [ORDI96].

Por esta razón, en muchas ocasiones el administrador de una máquina y del sistema de agentes otorgará privilegios adicionales a agentes móviles que pertenezcan a una autoridad y que provengan de un sitio específico. SAHARA es suficientemente versátil en este aspecto, por lo que será posible realizar la asignación en función de ambos parámetros, por ejemplo:

```
grant signedBy "Luis@156.35.31.66"  {
   permission FilePermission "/users/*", "read, write";
   permission SocketPermission "*", "accept, connect";
```

};

En este caso los privilegios especificados serán otorgados a los agentes de Luis siempre y cuando sus agentes provengan de la máquina 156.35.31.66[70].

12.3.10 Asignación de Permisos

Claramente se puede apreciar que un *CodeSource* (autoridad y/o direción de origen) puede coincidir con múltiples entradas en el fichero de políticas de seguridad, ya que el carácter "*" se permite y porque puede haber más de una ocurrencia del mismo.

Para saber cual será el conjunto de privilegios apropiados que le serán asignados a una autoridad, se utilizará el siguiente algoritmo:

Encontrar las llaves públicas si el código esta firmado.
Si la llave no es reconocida en el fichero de políticas de seguridad, hacer caso omiso de la llave y tratar el código como si no estuviera firmado o como no confiable.
Si las llaves son encontradas, o no se especifica ningún firmante entonces se deben encontrar todas la URLs asociadas en el fichero de seguridad para esas llaves.
Si una llave o una URL no es encontrada, se usan los permisos por defecto para código no confiable que suelen ser tan restrictivos como el administrador los haya asignado.

Si varias entradas son encontradas, entonces todos los permisos especificados en esas entradas son concedidos. En otras palabras, la asignación de permisos es aditiva, cada vez que se encuentre una entrada para una autoridad específica se irá añadiendo a los permisos que le fueron concedidos a esa autoridad previamente. Por ejemplo, si un agente firmado con la llave A obtiene el permiso X y un agente firmado por B obtiene el permiso Y, entonces el código firmado por A y B obtiene los permisos X y Y. Análogamente, si a la URL "http://uniovi.es" se le otorga el permiso X y a "http://uniovi.es/oviedo3" se le otorga el permiso Y, entonces un agente que provenga de "http://uniovi.es/oviedo3/arturop" obtendrá ambos permisos X y Y (asumiendo que los firmantes coinciden en caso de existir).

El significado exacto del valor de una URL en una entrada *CodeBase* depende de los caracteres del final. Un *CodeBase* con una terminación "/" se asocia con los archivos de clase que darán origen al agente (no ficheros JAR[71]) en el directorio específico. Un *CodeBase* con una terminación "/*" se asocia con todos los archivos, tanto con los ficheros de clase como archivos JAR, en ese directorio especifico. Un *CodeBase* con una terminación "/-" se asocia con todos los archivos, tanto con los ficheros de clase como archivos JAR, en el directorio y recursivamente todos los ficheros en subdirectorios contenidos en dicho directorio.

[70] Es posible solicitar que el agente sea originario de dicha dirección y que no haya viajado antes, para que no haya estado expuesto a ningún tipo de ataque en sus datos.
[71] Significa Java Archive o archivo de Java. Este formato de ficheros permite insertar todos los clases asociadas con un agente o una aplicación.

12.3.11 Soporte de múltiples ficheros de seguridad

En la política de seguridad por defecto del sistema, los permisos pueden ser especificados dentro de uno o más ficheros de configuración. Todos los ficheros de configuración especifican que permisos les son concedidos a agentes de diferentes orígenes.

Las localizaciones de dichos ficheros se especifican como valores de propiedades del sistema cuyos nombres son de la forma:

```
policy.url.n=URL
```

Donde 'n' es un número y URL es una especificación de URL. Por ejemplo, el fichero de políticas de seguridad por defecto del sistema está definido en el archivo de propiedades del sistema de la siguiente forma:

```
policy.url.1=file:${SAHARA.home}/security/system.policy
```

Se puede especificar cualquier número de URLs y todos los ficheros de seguridad serán cargados. El algoritmo comienza buscando policy.url.1, y continua incrementándose hasta que no encuentra un número consecutivo más. Si sólo se definen policy.url.1 y policy.url.3, la última nunca será encontrada. Obviamente para cada URL adicional habrá que especificar su ruta completa de localización.

12.3.12 Versatilidad para que los agentes compartan ficheros de forma protegida

Dentro de las principales ventajas de los agentes móviles se encuentra la realización de tareas en paralelo, para ello, a menudo es necesario que un agente procese información y la guarde y que el resto de los agentes accedan a esa información para continuar con el trabajo. Es decir provee seguridad al trabajo en grupo de los agentes.

Esta característica de permitir que los agentes puedan definir que privilegios otorgan a los ficheros comparten en la política de seguridad del sistema así como a quién le otorgan permiso es una virtud única que ningún sistema de agentes posee actualmente.

12.3.12.1 El método share

Cada agente puede determinar que ficheros desea compartir, si lo desea, usando el método *share* al que se le pasará como parámetro el nombre del fichero (incluyendo la ruta completa) de los archivos que haya creado y que desee compartir. El agente podrá usar los caracteres comodín[72] descritos anteriormente para poder especificar más de un archivo.

[72] Es decir, las terminaciones "/", "/*" y "/-".

El método *share* está sobrecargado por lo que adicionalmente existirán dos formas de poder especificar el resto de los parámetros. Para el primer caso habrá que señalar una variable que represente los privilegios que serán concedidos a ese fichero. Dicha variable contendrá tres números octales que al estilo *unix* representen los permisos de los diferentes tipos de usuarios sobre el fichero. Para los agentes remotos los números representan los permisos para la autoridad del fichero, para la máquina de donde proviene y para todos los agentes activos en el sistema respectivamente. Para los agentes locales el segundo número representa los permisos concedidos a los agentes del lugar al que pertenece dicho agente.

Al igual que en *unix* cada número octal, es una abreviación de su equivalente binario de tres bits **rwx**, en donde cada bit representa un tipo de permiso especifico. De izquierda a derecha se encuentran lectura (**r**ead), escritura (**w**rite) y ejecución (e**x**ecute). Si el bit está encendido representa que se tiene el permiso para realizar la actividad correspondiente. Por ejemplo un cinco (101), representa que se tienen permisos de lectura y ejecución.

Análogamente un 760, significa que la autoridad propietaria del agente gozaría de todos los privilegios sobre ese fichero. Los agentes que provengan de la misma máquina podrán leerlo y escribirlo, el resto de los agentes no tendrán ningún privilegio sobre él.

Para lograr que dichos privilegios surtan efecto, es necesario que el sistema de agentes identifique la autoridad y procedencia del agente que invoca el método. Después tendrá que editar el fichero de políticas de seguridad, en donde localizará todas las autoridades y direcciones correspondientes para asignar dichos privilegios. Es obvio suponer que puede haber más de una ocurrencia. Por último tendrá que ejecutar una orden de refresco del fichero de políticas de seguridad para que el nuevo fichero entre en vigor.

La segunda forma de usar el método *share* consiste en pasarle como parámetros además de la dirección del fichero, la autoridad y el *codebase* a quien le serán otorgados dichos privilegios, así como un número octal que represente los privilegios que le serán otorgados.

En este caso la tarea para el sistema de agentes es menos ardua, ya que editará el fichero de políticas de seguridad, pero en este caso sólo tendrá que buscar la ocurrencia que coincida con los datos de la autoridad que firma y de la dirección especificada en el *codebase*, una vez que la encuentre añadirá el permiso señalado y dará la orden de refresco del archivo de seguridad.

Este esquema permite a un agente poder otorgar privilegios sobre un fichero a un grupo de autoridades o bien a alguien específico, por lo que resulta más versátil y eficiente que otros modelos que habían utilizado un esquema similar (aunque más restrictivo) para compartir atributos, como en [NISH95a] y [NISH95b].

12.4 CUMPLIMIENTO DE PRIVILEGIOS

Una vez que los privilegios han sido asignados de forma deseada es necesario implementar una política que se encargue de verificar el cumplimiento de los permisos

asignados, y que asegure que ningún agente podrá realizar más tareas de las que le han sido permitidas en la política de seguridad del sistema.

12.4.1 Cumplimiento de los privilegios otorgados en los ficheros de políticas de seguridad del sistema de agentes

La unidad de protección y cumplimiento de privilegios básico dentro de la arquitectura se denomina **dominio de protección** que no es más que un conjunto de clases cuyas instancias dan origen a los agentes móviles y que gozan del mismo conjunto de permisos.

Un dominio esta identificado de forma única por su *codesource* (autoridad firmante y/o procedencia) el cual como he mencionado anteriormente encapsula dos características de los agentes que se ejecutan dentro de ese dominio: la dirección de procedencia y un conjunto de certificados para las llaves públicas que corresponden a las llaves privadas que firmaron todo el código en ese dominio.

Todas las clases/agentes firmadas por las mismas llaves y provenientes de la misma máquina se asocian al mismo dominio y por consiguiente tendrán los mismos permisos. Sin embargo, las clases que tienen la misma entidad firmante pero que especifican una dirección diferente en su *codesource* pertenecen a dominios distintos.

Un dominio de protección engloba también los permisos concedidos a los agentes de ese dominio, como es determinado de la política de seguridad que se encuentre en efecto en ese momento.

Los dominios de protección pueden clasificarse en dos categorías principales:

Dominios del sistema: estos dominios tienen acceso a todos los recursos protegidos del servidor como lo son el sistema de ficheros, la red y la pantalla.
Dominios de aplicación: son los que se crean para una autoridad específica y sus privilegios son determinados basándose en la política de seguridad del sistema.

Todo el código que forme parte de la plataforma en la que se implemente SAHARA se ejecutará dentro de un único dominio de protección del sistema, mientras que cada agente o grupo de agentes se ejecutará dentro de un propio dominio que será determinado por su *codesource.*

Existe una política de seguridad del sistema determinada en los ficheros de seguridad que son definidos por el administrador del sistema de agentes, los archivos especifican en cada entrada *grant* que nuevos dominios de protección deberán ser creados y que permisos le serán concedidos a cada dominio de protección.

El entorno de ejecución del sistema de agentes creará y mantendrá una relación de los agentes (las clases de código y sus instancias) a sus dominios de protección y de éstos a sus permisos, como se aprecia en la figura 12.4.1.1:

La seguridad en el entorno de ejecución estará definida por un objeto gestor de seguridad, podrá haber múltiples instancias de ese objeto[73] pero sólo una podrá estar en efecto en un momento dado.

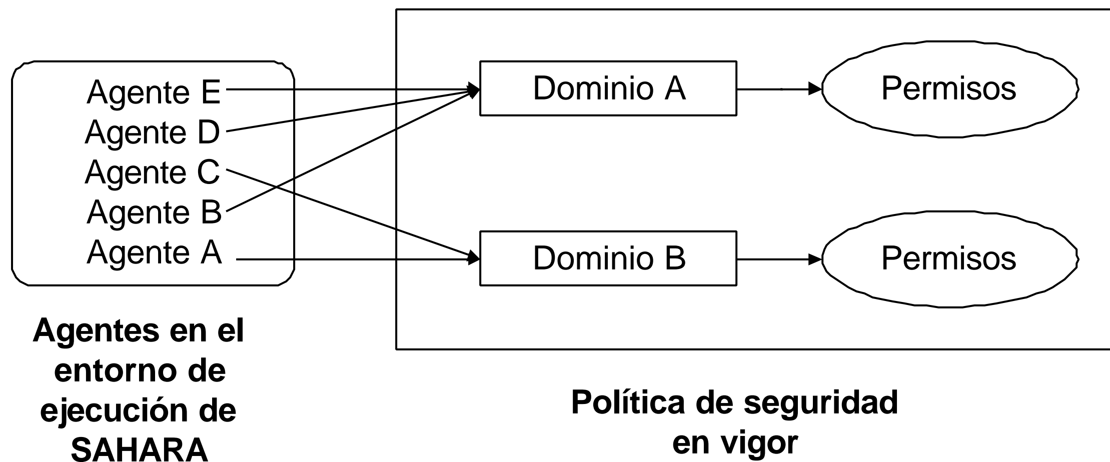


Figura 12.4.1.1 – Dominios de protección y permisos

El objeto gestor de seguridad actual será consultado por el dominio de protección cuando éste se encuentre inicializando sus permisos. El dominio de protección le pasará en un objeto su *codesource*, es decir, los certificados para las llaves asociadas a la autoridad que los firma y la dirección *codebase* si existiera. El objeto de seguridad evaluará la política global de seguridad (que es la suma de todos los ficheros de seguridad) y regresará un objeto apropiado de permisos que especifica los permisos concedidos a los agentes del *codesource* especificado.

La localización de la información origen utilizada por el objeto gestor de la seguridad depende de la política de implementación, como vimos anteriormente, normalmente se encuentran definidas en las propiedades del sistema.

Los dominios de protección son creados por demanda. Es decir, cada vez que un agente móvil llega a un nuevo destino, el sistema de agentes que lo recibe analiza los ficheros de políticas de seguridad para buscar una entrada de la autoridad o *codesource* que le corresponda, una vez que lo encuentra obtendrá los permisos asignados para dicho *codesource* y entonces creará un nuevo dominio de protección en donde asocie el agente en cuestión con los privilegios especificados para dicho *codesource.*

Si llega otro agente que provenga del mismo sitio y sea de la misma autoridad, es decir que sea de un mismo *codesource,* el sistema de agentes antes de crear otro dominio de protección debe buscar si existe un dominio que ya haya sido asignado a dicho *codesource*, si lo encuentra como en este caso, solo asociará al agente que acaba de llegar con el dominio de protección que había sido previamente creado para que surtan efecto los permisos de dicho dominio sobre el agente.

[73] Cada una podrá definir una política de seguridad distinta

Es importante resaltar que se crea un dominio para cada autoridad aunado a su procedencia (si se especifica), por lo que los agentes que hayan sido creados en un mismo servidor pero que pertenezcan a diferente autoridad tendrán dominios de protección distintos. De igual forma los agentes firmados por una misma autoridad pero que especifiquen una dirección de procedencia distinta en la entrada *grant* del archivo de políticas de seguridad también tendrán un dominio de protección distinto.

Por eso, es posible especificar en la entrada *grant* de este fichero sólo la entrada de la autoridad que firma o bien sólo la dirección de procedencia, para que agentes que pertenezcan a dicha autoridad o que provengan del sitio especificado sean asociados a un mismo dominio de protección, y por consiguiente gocen de los mismos privilegios.

Por último para los agentes móviles locales se creara un dominio de protección para cada autoridad local en el sistema, dicho dominio será creado basándose en su *codebase*, todos los agentes que sean creados a partir del mismo *codebase* serán asociados a un único dominio de protección que englobará los permisos concedidos a dicha autoridad local.

12.4.2 Resolución de permisos en invocaciones a dominios de protección distintos

Un agente puede ejecutarse dentro de un dominio de protección individual o puede involucrar tanto al dominio de aplicación como al dominio del sistema.

Por ejemplo, un agente que imprime un mensaje tendrá que interactuar con el dominio del sistema, que es el único punto de acceso a ese recurso[74]. Es este caso es crucial que en ningún momento el dominio de aplicación gane permisos adicionales por llamar al dominio del sistema. En la situación contraria el dominio del sistema podría invocar un método de un agente que pertenece al dominio de aplicación, una vez más es crucial que los derechos efectivos de acceso sean los mismos que los derechos concedidos al dominio de aplicación.

En otras palabras, un dominio **menos** poderoso no puede ganar permisos adicionales como resultado de llamar a un dominio más poderoso; mientras que un dominio más poderoso debe perder su poder cuando llame a un dominio menos poderoso.

Esta argumentación sobre los permisos que un agente debe tener cuando se involucra con dos dominios de protección naturalmente se generaliza a un agente que atraviesa múltiples dominios de protección. De aquí se origina una regla simple y prudente para calcular los permisos que es la siguiente: “El permiso de un agente en ejecución es la intersección de todos los permisos de todos los dominios de protección que atraviese las llamadas de dicho agente”.

Esta regla se aplicará como norma definitiva, no obstante, un agente podrá gozar de privilegios especiales por un breve lapso de tiempo otorgados por el dominio del

[74] Como se explicó en la sección previa, el dominio del sistema es el único que tiene acceso directo a todos recursos del servidor.

sistema o bien por el dominio de otro agente, donde al resultado de la regla anterior habría que añadir el privilegio especial mencionado.

12.4.3 Cumplimiento de las asignaciones

Como expliqué anteriormente, cada sistema de agentes contará con una base de datos llamada SAHARA/Security/**DBResources** en donde se especifican las cantidades máximas de memoria, espacio en disco y agentes hijos permitidos para cada sistema remoto y para cada autoridad remota.

Existirá un objeto en memoria **DBResourcesFree** que contenga las cantidades disponibles de cada Servidor de agentes remoto que este incluido en *DBResources*, este objeto estará asociado con su dominio de protección existente. El objeto será creado cuando el primer agente de un sistema remoto llegue, es decir justo después de que sea creado su dominio de protección.

Todos los dominios de protección que estén basados en la dirección remota del sistema de agentes (*codebase*) tendrán asociado este objeto *DBResourcesFree* que indicará la cantidad de recursos disponibles que quedan para dicho sistema remoto. Este tipo de objeto normalmente se les aplicará a los dominios de protección que se crearán de los servidores que pertenecen a una misma región.

Para los dominios de protección asociados a las autoridades locales de sistema y que están basados sólo en el *codebase* no se les asigna el objeto *DBResourcesFree* debido a que no se les imponen límites para el consumo de recursos locales. Se supone, que si el administrador abre una cuenta para una nueva autoridad local, es porque tiene un grado de confianza suficiente para permitirle usar y consumir los recursos del sistema.

Para los agentes cuyo dominio de protección está basado en su autoridad y en la dirección de donde provienen, se considerará ambos valores (autoridad y dirección) para asociar el objeto *DBResourcesFree* al dominio de protección que previamente haya sido creado, considerando los valores que le han sido asignados a esa autoridad y de ese servidor en la base de datos *DBResources.*

Para los agentes cuyo su dominio está basado únicamente en la autoridad de donde provienen, el objeto *DBResourcesFree* se asociará de igual forma con su dominio de protección, el cual esta basado en la autoridad. La única diferencia es que los valores otorgados en *DBResources* estarán definidos basándose en su autoridad y no en el servidor de donde provengan. Esto se puede apreciar en la figura 12.4.3.1.

De esta forma, cada vez que un agente llegue de un sistema de agentes remoto o bien cuando un agente se duplique, se actualizarán los valores correspondientes del objeto *DBResourcesFree* que se encuentra asociado con su dominio de protección correspondiente. De esta manera, los agentes provenientes de un mismo sistema o de una misma autoridad podrán navegar sin ningún problema dentro del sistema remoto mientras ejecuten las tareas permitidas dentro del dominio de protección en el que se encuentran y no agoten los recursos que le han sido otorgados al servidor de donde provienen o a la autoridad a la que pertenecen que se especifican en *DBResources.*.

Por último, los agentes provenientes de servidores desconocidos también serán contabilizados de igual manera, salvo que en vez de que exista un registro en **DBResources** para dicho sistema o autoridad, se usará el valor por defecto para servidores no autentificados, el cual podría ser un poco menor o muy restrictivo, dependiendo de la política de seguridad que se encuentre en vigor.

Así pues, un sistema de agentes autorizará las cantidades solicitadas por la asignación del agente siempre y cuando dichas cantidades no excedan los límites disponibles fijados para el sistema de agentes del que proviene o para la autoridad a la que pertenece.

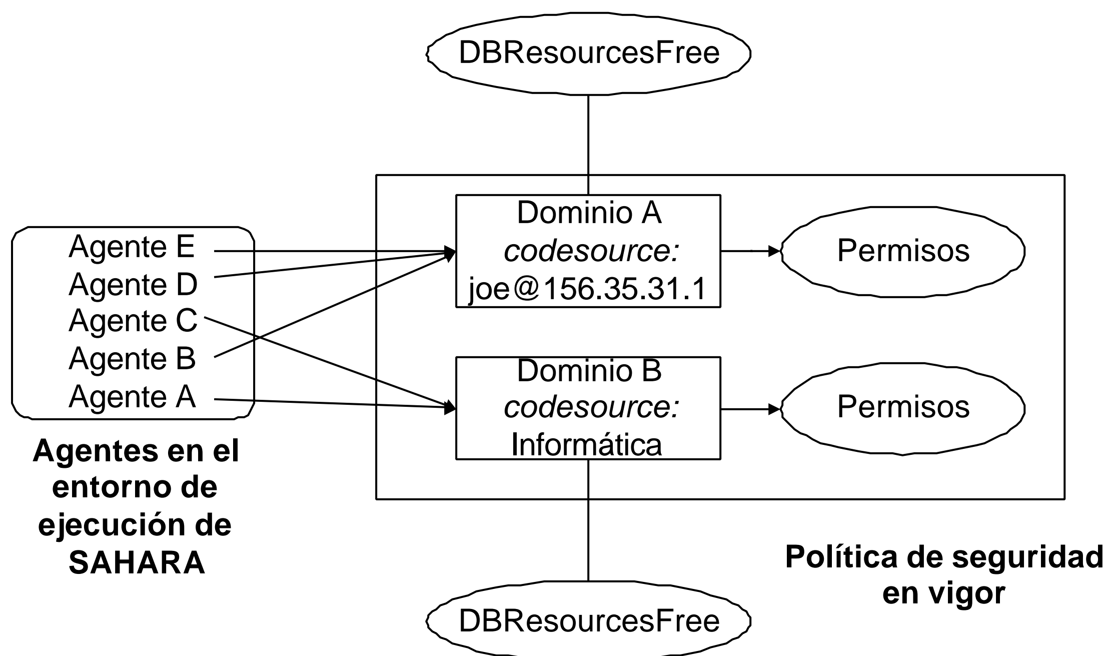


Figura 12.4.3.1 – Dominios de protección y Asignaciones

12.4.4 ¿Qué se logra con la Autorización y el Cumplimiento de Privilegios?

El agrupamiento de agentes en dominios de protección nos permite asegurar que agentes de cualquier dominio que no sean del sistema no podrán tener conocimiento de la existencia de agentes en otro dominio de aplicación. Esta partición es lograda gracias a una resolución y carga de clases/agentes cuidadosa, usando diferentes cargadores de clases para dominios diferentes.

Si se puede asegurar que ningún agente podrá ver a otros agentes (sin la ayuda del servidor de agentes) y que ningún agente puede acceder a los recursos de otro, se puede asegurar que no habrá ataques de agentes contra otros agentes.

Igualmente la existencia de las asignaciones asociadas a un dominio de protección, nos permiten controlar los recursos que consumen los agentes de una máquina determinada

o de una autoridad específica. Mientras los recursos sean contabilizados y controlados no podrán existir ataques de agentes que provengan de un servidor o autoridad maliciosa que pretendan agotar los recursos de un grupo de máquinas y con ello corromper la integridad de una red de servidores de agentes.

De esta forma SAHARA da solución a estos dos tipos de ataques, a los cuales son vulnerables muchos sistemas de agentes móviles.

Capítulo XIII
SAHARA: PROTECCION DE LA TRANSMISIÓN DE AGENTES E INFORMACIÓN CONTRA TERCERAS PARTES

Otro ataque que pueden sufrir los sistemas de agentes móviles es el espionaje, alteración o falsificación de agentes o de la información que éstos llevan consigo. Para evitarlo la arquitectura de SAHARA utiliza el protocolo de transferencia SSL que se ha convertido en un estándar para la transmisión de información en Internet. Este protocolo garantiza básicamente integridad, autenticación y confidencialidad (privacidad).

Para solucionar este problema, no se pensó en reinventar la rueda creando un nuevo protocolo de transferencia que se saliera de los estándares y que después dificultará las tareas de integración con otros sistemas. Por el contrario se pensó en analizar los protocolos actuales y seleccionar el que fuera el mejor, para implementarlo e integrarlo a esta arquitectura de tal forma que SAHARA pudiera comunicarse de forma estándar con otros sistemas.

El protocolo SSL autentifica los servidores que se van a conectar antes de iniciar la transmisión. Sólo dos sistemas en este año han decidido incorporar SSL como protocolo para la transmisión de agentes, ARA[PEIN98] y Concordia [WALS98], no obstante, no gozan de una arquitectura integral como la que ofrece SAHARA. Algunos otros sistemas utilizan protocolos menos seguros y eficientes. (Por ejemplo D'Agents, Aglets).

13.1 PGP O SSL

Los dos protocolos más usados actualmente para transmitir información encriptada de una forma segura en la red son PGP(Pretty Good Privacy) y SSL(Secure Sockets Layer).

El protocolo de muy buena privacidad PGP puede ser usado para la utilización de firmas digitales y para encriptación. PGP es un programa individual que permite la transmisión segura de correo electrónico y que es ampliamente usado.

PGP encripta un fichero o un mensaje de correo usando el algoritmo IDEA y una llave secreta escogida aleatoriamente, encripta la llave secreta usando el algoritmo RSA de llave pública y la llave pública del receptor. PGP opcionalmente añade una firma digital computando un valor *hash* criptográfico MD5 del fichero o del mensaje de correo y encriptando el valor *hash* con la llave privada del emisor.

Aunque PGP esta orientado al uso interactivo de correos electrónicos puede ser también usado en un sistema de agentes, no obstante no presenta todas las ventajas de SSL que fue un protocolo hecho para la transmisión de información entre dos servidores.

SSL es un protocolo de seguridad que provee comunicaciones secretas y confiables entre dos aplicaciones. La manera de comunicación entre el cliente y el servidor esta diseñada para prevenir la falsificación de mensajes o el espionaje de información.

El protocolo está compuesto de dos capas. En el nivel más bajo, sobre la cima de algún protocolo de transferencia confiable (por ejemplo TCP/IP), se encuentra el protocolo SSL de registro, el cual es usado para encapsular varios protocolos de más alto nivel. Uno de esos protocolos encapsulados es el protocolo de saludo de SSL, que permiten al servidor y al cliente autentificarse mutuamente y negociar un algoritmo de encriptación y llaves criptográficas antes de que el protocolo de la aplicación transmita o reciba el primer byte de datos.

Una ventaja de SSL es que es un protocolo independiente de la aplicación. Un protocolo de nivel más alto puede establecerse sobre el protocolo SSL transparentemente. El protocolo SSL provee una seguridad en la conexión que tiene tres propiedades básicas:

- La conexión es privada. La encriptación es usada después de un saludo inicial que define una llave secreta para el envío de información. Esto protege la confidencialidad de los datos
- La identidad del punto que se conecta puede ser autentificado, se usa criptografía asimétrica o de llave pública para lograrlo.
- La conexión es confiable. El transporte de mensajes incluye una verificación de la integridad del mensaje para asegurar que nadie la ha modificado.

Debido a sus propiedades SSL resulta ser idóneo porque permite no sólo la transmisión segura de los agentes y sus datos si no también la autenticación mutua de sus servidores.

13.2 LAS VENTAJAS DE SSL

Como SSL se conecta dentro de una aplicación en la capa de *socket*, y puede convenientemente situarse debajo de otros protocolos como HTTP, RPC, IIOP y RMI, esencialmente proporcionándoles seguridad a estos protocolos. Además al conectarse en el nivel de *sockets*, no requiere ningún conocimiento de seguridad especial o código

relacionado con seguridad en los niveles más altos de la aplicación, haciéndolo de alguna forma más fácil de implementar que otras soluciones.

La posibilidad de autentificar a los servidores que participan en la conexión les permite a los servidores protegerse contra ataques de otras máquinas y elimina la posibilidad de falsificación de servidores.

PGP no ha sido considerado [NETWa] porque su protocolo se asemeja más a un formato de mensajes que a un protocolo, por ejemplo carece de protección contra réplicas; mas aún, su modelo de llavero (anillo de llaves) para la certificación de llaves públicas parece poco apropiado para grandes poblaciones dinámicas.

Otra razón es que PGP requiere un mayor esfuerzo para integrarlo a una aplicación basada en sockets que SSL, el cual es más eficiente debido a su concepto de sesión, que permite intercambiar los datos a través de transferencias individuales entre los mismos servidores, reduciendo con ello las costosas computaciones de llave pública.

Finalmente dentro de los objetivos que han sido considerados en la creación de SSL [FREI96] encontramos:

1. *Seguridad criptográfica:* que es usada para establecer una conexión segura.
2. *Interoperabilidad:* cualquier aplicación que utilice el protocolo podrá intercambiar parámetros criptográficos sin el conocimiento del código del otro.
3. *Extensibilidad:* el protocolo se podría ampliar para incorporar nuevos métodos que fuesen necesarios.
4. *Relativa eficiencia:* las operaciones criptográficas tienden a ser más altamente intensivas con el CPU, particularmente las operaciones de llaves públicas. Por esta razón incorpora un esquema de sesión óptimo que reduce el número de conexiones que necesitan ser establecidas desde el principio.

13.3 DESCRIPCIÓN DEL PROTOCOLO SSL

La confidencialidad, autenticación e integridad que proporciona el algoritmo son logradas por una “especificación de cifrado de mensajes” que no es más que el resultado de una combinación de algoritmos criptográficos usados por una conexión SSL específica.

Cuando una conexión SSL es iniciada, no lleva ningún mecanismo de protección. Sin embargo, la seguridad se provee en el tiempo que los datos de la aplicación son enviados a través de la conexión. Antes de enviar los datos, los servidores de agentes deben expresar sus preferencias sobre la especificación de cifrados de mensajes que pueden ser usados en la comunicación. El código de SSL realiza una negociación como parte de la preparación del envío de los datos. La conexión será establecida si alguna especificación de cifrado de mensajes es soportada por ambas partes de la conexión. Si no existe tal especificación o no se llega a un acuerdo en la negociación la conexión no será establecida.

SSL es un protocolo de capas. En cada capa los mensajes pueden incluir campos para indicar la longitud, descripción y contenido. SSL toma los mensajes a ser transmitidos, fragmenta los datos en bloques, los comprime, los encripta y luego transmite los resultados. Los datos recibidos son desencriptados, verificados y descomprimidos para ser entregados a los clientes de capas más altas.

13.3.1 Los estados de sesión y conexión

Una sesión SSL puede incluir múltiples conexiones seguras Adicionalmente los grupos pueden tener sesiones múltiples simultáneas.

Una sesión SSL tiene estados. Lógicamente, los estados son representados como el estado actual de ejecución y como el estado pendiente (de ejecución). Adicionalmente son mantenidos por separado estados de lectura y escritura. Cuando un cliente o servidor envía una especificación de cifrado de mensajes, el estado pendiente de escritura se copia hacia el estado actual de escritura. Una operación similar se realiza con el estado de escritura al recibir un mensaje.

13.3.2 La capa de registro

La capa de registro de SSL recibe datos de capas más altas en bloques no vacíos de tamaño arbitrario.
La capa de registro fragmenta los bloques de información en registros de *SSLPlainText* de 2^14 bytes o menos. Todos los registros son comprimidos usando el algoritmo de compresión definido en el estado de sesión actual.

13.3.3 El protocolo de saludo

Los parámetros criptográficos del estado de sesión son producidos por el protocolo de saludo, el cual opera en la cima del protocolo de registro de SSL.

La especificación de cifrado de mensajes usada para el intercambio de mensajes es establecida por un proceso de negociación entre el cliente y el servidor SSL, el cual es llamado "saludo". El objetivo de este proceso es crear una sesión, que puede proteger varias conexiones a través del tiempo.

Cuando un cliente y servidor inician la comunicación se ponen de acuerdo en cuanto a la versión del protocolo, algoritmos criptográficos seleccionados, técnicas de encriptación de llaves públicas para generar los secretos compartidos. Todo esto es hecho por el protocolo de saludo que se resume de la siguiente manera: el cliente envía un mensaje de *hello* al cual el servidor debe responder con un mensaje *hello*. El cliente y el servidor *hello* establecen los siguiente atributos: versión del protocolo, identificador de sesión, series cifradas y método de compresión. Adicionalmente se general dos valores aleatorios y son intercambiados.

Enseguida de los mensajes *hello* el servidor enviará su certificado, si este tiene que ser autentificado. Y éste a su vez podrá solicitar un certificado del cliente si esto es apropiado a la serie cifrada seleccionada. Entonces, el servidor enviará el mensaje del servidor *hello* completo indicando que ésta fase esta completa. El servidor entonces esperará por la respuesta del cliente. Si el servidor solicitó un certificado el cliente debe enviar el mensaje del certificado que inclusive puede ir firmado digitalmente.

En este punto la especificación de cifrado de mensajes es enviada por el cliente. El cliente inmediatamente envía el mensaje de finalización bajo los nuevos algoritmos, llaves y secretos. En respuesta el servidor le enviará su propia especificación de cifrado de mensajes y envía su mensaje de finalización bajo la nueva especificación. En ese punto el saludo esta completo y puede comenzar el intercambio de datos

El proceso se describe mediante el siguiente esquema de flujo de datos:

CLIENTE	INFORMACIÓN	SERVIDOR
ClientHello	→	
		ServerHello [75]Certificate*
	←	ServerKeyExchange* CertificateRequest* ServerHelloDone
Certificate* ClientKeyExchange		
CertificateVerify* [ChangeCipherSpec] Finished	→	
	←	[ChangeCipherSpec] Finished
Application Data	←→	Application Data

Después de que el "saludo" ha sido completado se puede acceder a los atributos de la sesión usando el método *getsession*. El saludo es completamente transparente en la mayor parte de los casos, pero puede ser iniciado explícitamente.

13.3.4 Atributos usados del servidor de agentes de SAHARA

Para implementar el protocolo dentro de SAHARA es necesario que cada servidor de agentes tenga su propia llave pública y privada. Como vimos anteriormente la estructura *NameAgentSystem* que define los atributos de identificación de un sistema, contiene:

```
String Authority          // propietario del sistema de agente
Int AgentSystemId         // identificador único del sistema de agentes
String IP                 // dirección IP del sistema de agentes
```

[75] El * indica que es opcional o que son mensajes dependientes de la situación que no siempre son enviados.

Por lo que se utilizarán las llaves de la autoridad a la que pertenece el sistema de agentes para establecer la conexión. Dichas llaves al igual que las de todas las demás autoridades que se encuentran en el sistema tendrán su propio certificado (almacenado en el *keystore)* el cual podrá ser utilizado para el protocolo de saludo, en caso de que sea requerido por el servidor remoto.

El acceso a las llaves y al *keystore* solo se realizará la primera vez mientras se establece la conexión, posteriormente sólo se iniciaran sesiones cada vez que exista una transferencia de agentes.

El protocolo SSL siempre será iniciado para la comunicación entre los dos servidores sin importar si éstos pertenecen o no a una misma región. Es decir, los servidores de agentes que transmitan agentes siempre utilizarán el protocolo para garantizar un envío seguro. Si el servidor de agentes no pertenece a una región, adicionalmente el agente será firmado, con lo que habrá una protección contra grupos terceros de dos niveles, una protección mediante el agente firmado y la otra la que brinda el protocolo al encriptar los datos.

Si el servidor de agentes pertenece a la misma región que el servidor al que va a ser transmitido el agente, entonces sólo mediante el protocolo SSL se protege la transmisión de los datos ya que los agentes no serán firmados.

13.4 ¿QUÉ SE LOGRA MEDIANTE EL USO DE SSL EN LA TRANSMISIÓN?

Básicamente el uso de SSL le proporciona a la arquitectura SAHARA la posibilidad de emplear el concepto de región para gestionarlo como un ente dentro del cual el nivel de confianza es grande y no se requieren verificaciones de seguridad adicionales, ya que permite la transmisión segura de agentes sin necesidad de ser firmados.

Por otra parte el protocolo es pieza fundamental para la protección de los servidores contra ataques de grupos terceros. Es decir al utilizar el protocolo resulta imposible la falsificación de servidores, agentes o autoridades, ya que todo servidor de agentes como elemento fundamental del desarrollo de una aplicación basada en agentes móviles es autentificado antes de iniciar cualquier interacción con cualquier otro servidor remoto.

De igual forma no es posible el robo o espionaje de agentes y datos ya que cada sesión iniciada por los servidores es codificada. La información que se pudiera extraer de dicho enlace carece de sentido para cualquier otro que no conozca las llaves o la especificación de cifrado por lo que el enlace se considera completamente privado.

Por último SSL proporciona versatilidad en la implementación ya que es independiente de la plataforma y puede ser montado sobre cualquier protocolo de transporte de red subyacente.

Capítulo XIV
SAHARA: PROTECCIÓN DE AGENTES MÓVILES CONTRA ATAQUES DE SERVIDORES MALICIOSOS

Quizá la más ardua labor en la protección de los sistemas de agentes móviles consiste en proteger a los agentes de un posible servidor malicioso ya que normalmente el servidor tiene acceso a todas las partes de un agente[76] y además tiene un control total sobre él.

Si el servidor es capaz de modificar el código puede alterar el comportamiento futuro del agente, mientras que si modifica sólo sus datos alterará los resultados que el agente genere.

Actualmente se puede solucionar satisfactoriamente el primer problema ya que el código del agente no cambia y puede ser firmado digitalmente, mientras que la protección de los datos es más complicada porque estos irán cambiando constantemente con lo que sólo se pueden proporcionar solucionar parciales al problema o bien soluciones totales pero difíciles de implementar.

14.1 PROTECCIÓN DEL CÓDIGO

Afortunadamente, como vimos en la sección 11.3.1, los agentes antes de ser enviados serán firmados digitalmente. Es decir, el código del agente va firmado por la autoridad a la que representa en principio para ser autentificado y también para ser protegido.

Si el código de un agente es modificado por un servidor malicioso y luego enviado a otro servidor, al llegar al llegar al nuevo servidor el agente será rechazado porque al comprobar la firma de la autoridad a la que pertenece ya no será válida. De esta forma es imposible alterar el comportamiento del agente móvil, o en su defecto en caso de ser

[76] Es decir, sus tres estados, su estado de código, su estado de datos y su estado de ejecución.

modificado será detectado inmediatamente con lo que el agente no podrá causar daños en otros servidores con su comportamiento alterado.

Las firmas digitales son fáciles de gestionar y garantizan una total protección del código móvil que no será alterado.

14.2 PROTECCIÓN DE LOS DATOS

Desgraciadamente para proteger a los datos no se tiene la misma suerte, ya que estos no pueden ser firmados digitalmente cuando salgan del servidor de origen porque el mismo agente los irá modificando en su recorrido. Tampoco es posible que el agente transporte una llave con la que los pueda firmar porque esa llave también podría ser usada por el servidor malicioso para acceder o falsificar la información, ya que el servidor tiene acceso a todas las variables del agente.

Para dar solución a este problema tan complejo, SAHARA implementa inicialmente un **proceso de firmado secuencial usando la firma de la autoridad del servidor local**. Es decir, cada vez que el agente va a migrar el servidor local firma el estado de datos del agente afirmando con ello que no ha distorsionado sus datos. De esta forma se le adjudica responsabilidad a las autoridades dueñas de los servidores para poder penalizarles en caso de que firmen datos alterados[77].

A pesar de las garantías expuestas anteriormente, no se puede garantizar una protección total por lo que se usan diversas técnicas que garantizan la migración de los agentes a sitios 100% seguros, o bien si es necesario que el agente navegue sin límites en Internet, las técnicas de SAHARA permitirán saber si los resultados obtenidos por el agente son confiables.

14.2.1 Protección de los datos mediante navegación en redes seguras

Quizá la solución más sencilla para evitar el ataque de un servidor malicioso a un agente es el cuidar que el agente nunca este al alcance de un servidor malicioso. Para lograrlo cada servidor constará de un fichero *SAHARA/config/roaming.cfg* en donde se especifiquen todas las redes, subredes y ordenadores a donde los agentes que lleguen a ese sistema podrán enviarse.

Cada sistema de agentes incluirá en el archivo sólo aquellos sistemas en quienes tiene plena confianza de que no le harán ningún daño a sus agentes ni los enviará a servidores indeseables, de esta forma un grupo de universidades o empresas pueden definir una red fiable de navegación.

Para facilitar el formato y lograr el objetivo del fichero *roaming.cfg* que es permitir la especificación de una red, una subred o un ordenador en específico, bastará con

[77] Se busca una analogía con la vida real en donde se penaliza a cualquier autoridad que firme fasilmente un documento.

especificar la dirección de la red, o del ordenador en especifico. Para el caso de las subredes, se pondrá en intervalo de las direcciones deseadas separadas por un '-'.
Para el caso de redes enteras u ordenadores específicos se puede sustituir la dirección IP por su nombre correspondiente.

Así pues el contenido de dicho fichero podría ser:

(El ordenador especifico)
(Todos los dispositivos de 156.35.95.129 a 156.35.95.200)
(Todos los ordenadores de la red 156.35.94.*)
pinon.ccu.uniovi.es (El ordenador pinon)

Para el caso de sistemas de agentes con una gran cantidad de agentes móviles, el formato del fichero fácilmente puede ser cambiado a un sistema parecido al que tienen las tablas de encaminamiento de los sistemas UNIX para definir el encaminamiento que siguen los mensajes IP para llegar a su destino. Dicho formato especifica una red y una máscara para cada registro en el fichero.

La red define el grupo de direcciones a donde se podrá enviar al agente, mientras que la máscara se utilizará para tomar un subconjunto de esa red y definir una subred de envío[78]. Las direcciones de dispositivos únicos podrían indicarse directamente en vez de una dirección de red, solo cambiando el último número que es cero por el número del dispositivo apropiado.

Esta forma de protección es perfecta y fácil de implementar, desgraciadamente no contribuye a uno de los fundamentos que originaron esta tecnología que es la de *sistemas de agentes móviles abiertos*. Para muchas aplicaciones se requiere que los agentes naveguen ilimitadamente en Internet, fuera de redes confiables, por lo que se implementan algunas técnicas para garantizar la integridad de los datos.

14.2.2 Protección de datos mediante técnicas de duplicación de datos

Esta técnica tiene como objetivo verificar si los datos del agente han sido manipulados y modificados, con lo que se detectará cuando los resultados no son fiables.

La utilidad de esta técnica y la próxima es sobre todo para aquellas aplicaciones de comercio electrónico en donde se envía a un agente a navegar con un itinerario fijo a que busque el mejor precio de un producto o servicio con las características especificadas a la red.

El ataque típico en este tipo de aplicaciones es que un servidor malicioso identifique cual es el producto/servicio buscado y modifique el precio de aquellos competidores que tengan un precio inferior a él, o bien que asigne un precio un poco inferior a los de los competidores para que dicho servidor malicioso aparezca como el del mejor precio.

Para solucionar este problema se hace lo siguiente:

[78] Para mayor información consulte [HUNT94]

Suponiendo que el agente sigue un itinerario de servidores S= $s_1,s_2,......,s_n$. Entonces tendrá que añadir una casilla más al vector de datos en donde guardara los resultados buscados y al vector del itinerario, los vectores llegarán hasta s_{n+1}.
En la primera casilla del vector de resultados del agente se define un valor ficticio que corresponda a un servidor ficticio muy por debajo del valor esperado.
Se hace una copia en el servidor local del valor ficticio agregado.
Se envía el agente a navegar.
Al regresar del itinerario se verifica que el valor más bajo sea el valor asignado al servidor ficticio y que corresponda al valor guardado en el servidor. Si no es el más bajo o no coincide al valor guardado significa que el agente fue alterado y sus resultados no son fiables.
Si el valor más bajo corresponde con el que se almacenó en el servidor, el resultado deseado por nosotros es el segundo más bajo.

14.2.3 Protección de datos mediante técnicas de duplicación de agentes móviles

La desventaja de la técnica anterior es que debe ser implementada directamente por el programador ya que el sistema de agentes no podría realizar estas verificaciones directa e independientemente.

Esta técnica obtiene los mismos resultados que la anterior. Sin embargo, definido un itinerario y un vector de resultados el sistema de agentes puede encargarse automáticamente del trabajo de verificación. La única condición que debe cumplirse es que dentro del itinerario definido solo exista un servidor malicioso y que éste no eliminará al agente. Al igual que la técnica anterior solo permite descubrir si el agente ha sido alterado.

Supongamos que el itinerario que ha seguir será $S= s_1,s_2,......,s_n$. Se programan dos agentes A_1 y A_2 donde A_1 seguirá el itinerario S y A_2 viajará a través de S^{-1}. Asumamos que el servidor Si es malicioso y que Sj es el servidor que posee mejor precio.
Consideremos en principio el caso $j < i$. A_1 encontrará primero el mejor precio (Sj) y cuando llegue a Si, su memoria del mejor precio será alterada. Cuando A_1 regrese con el resultado, éste informará que Si es el servidor que tiene el mejor precio o bien que Sk tiene el mejor precio donde $k > i$ si el servidor malicioso no asignó un precio inferior que aquellos que el agente se pudiera encontrar en el resto del recorrido.

A_2 por el otro lado, encontrará el mejor precio después de visitar el servidor malicioso. Esté a su regreso al servidor proporcionará el precio mínimo correcto suponiendo que el servidor malicioso no realizó otro tipo de ataque, ya que dicho servidor no logró engañarlo asignando un precio menor.

Cuando A_2 regrese, el servidor de agentes al ver que los datos no coinciden determinará que estos datos no son fiables. El usuario podrá decidir si confía en el menor precio proporcionado o si decide anular esa búsqueda e implementar otra.

Una desventaja que presenta esta técnica es en caso $i = j$. Cuando esto ocurre el servidor malicioso puede modificar su precio para situarlo justo por debajo del valor del segundo

precio más barato modificando su precio que era inferior para convertirlo en el más barato pero por la mínima diferencia. Ante esta situación no se tiene alternativa, aunque ciertamente el daño no afecta en nada a los resultados del agente.

14.2.4 Protección de datos mediante códigos de autenticación de resultados parciales

Esta es una técnica criptográfica que en gran medida resuelve en gran medida la protección de los datos ya que puede aplicarse a cualquier tipo de aplicación, los agentes podrán proteger íntegramente su estado de código. Está técnica será implementada con la primera versión de SAHARA.

La idea de los códigos de autenticación de resultados parciales (PRAC de sus siglas en inglés) es similar a los códigos de autenticación de mensajes (MAC). Pero en vez de autentificar los orígenes del mensaje, se demuestra la autenticidad de los resultados parciales que un agente obtiene come resultado de ejecutarse en un servidor.

La propiedad que los PRACs aseguran es una perfecta integridad al avanzar e el itinerario. Si un agente móvil visita una secuencia de servidores $S= s_1,s_2,......,s_n$, y el primer servidor malicioso es *Sc*, ninguno de los resultados parciales generados en los servidores *Si* donde $i < c$, puede ser falsificado.

Los PRAC nos aseguran la integridad total de los resultados hasta antes de que el agente se tope con un servidor malicioso. Para lograr la protección nosotros enviamos al agente con una lista de claves secretas PRAC, una clave por cada servidor a ser visitado. El complemento de esas claves secretas requerido para desencriptar la información queda dentro del servidor de forma protegida.

Antes de que el agente migre al siguiente servidor éste resume los resultados parciales de su trabajo en ese servidor en un “mensaje” que será retornado al servidor de origen del agente. Para proporcionar integridad, un MAC es generado en este mensaje, usando la clave secreta asociada con el servidor actual; dicho mensaje con su MAC respectivo constituyen la PRAC. La diferencia crítica entre los MACs y los PRACs es que después de que un PRAC es generado, el agente tiene el cuidado de borrar la clave PRAC asociada con el servidor actual antes de migrar al siguiente servidor. El funcionamiento de esta técnica se describe en la figura 14.2.4.1.

Una vez que esta clave asociada es borrada ninguna entidad o servidor malicioso será capaz de obtener la información codificada mas que el servidor original que tiene las llaves complementarias logrando una importante propiedad de seguridad: *los resultados parciales de los servidores honestos predecesores no pueden ser modificados.*

El PRAC no tiene que ser enviado inmediatamente, el agente puede llevarlo consigo como parte de su estado de código y enviarlo cuando las condiciones le favorezcan (cuando haya un ancho de banda apropiado o cuando tenga permiso para hacerlo).

Aunque el agente viaje temporalmente con el PRAC un servidor malicioso no puede acceder a los datos porque requeriría de la llave complementaria de la cual carece, lo

único que podría lograr sería alterar los resultados parciales de ese agente en dicho servidor malicioso pero nunca lograría alterar los resultados obtenidos en servidores anteriores.

Cuando el servidor origen reciba los mensajes de los agentes él podrá aplicar las claves complementarias para obtener los resultados parciales del agente en cada servidor y en caso de que los datos hayan sido alterados por un servidor malicioso podrá identificar en que momento los datos fueron modificados e inclusive identificar al servidor malicioso.

La única restricción que tiene esta técnica es que debe tener un itinerario fijo. O bien que el número de servidores que puede visitar esta en función del número de llaves secretas que posea. Esta desventaja puede ser fácilmente solucionada al realizar la siguiente mejora.

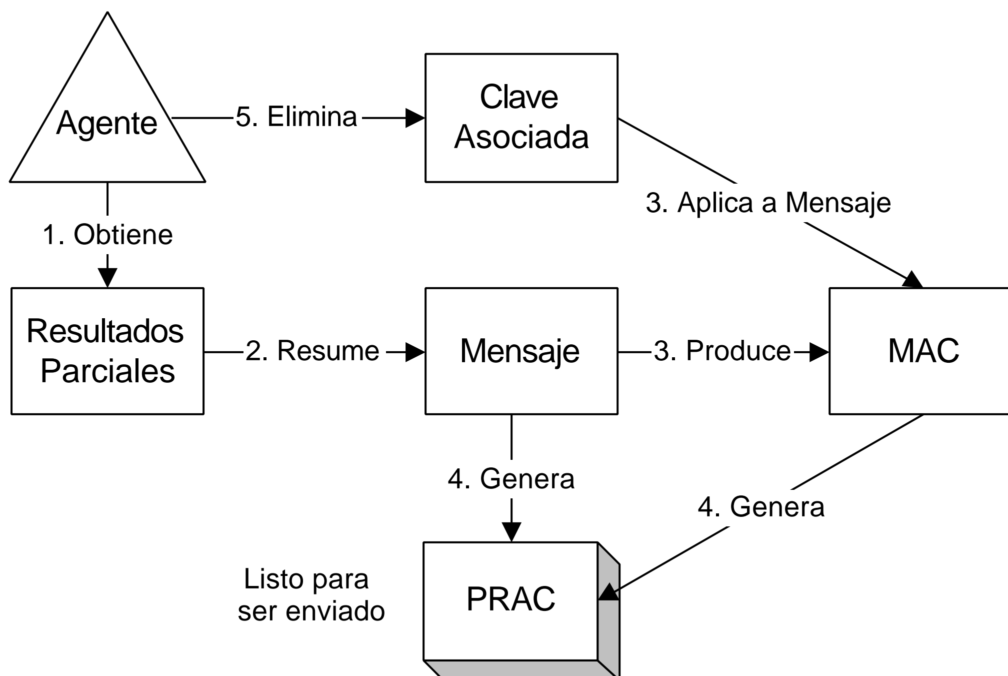


Figura 14.2.4.1 – Generación de códigos de autenticación de resultados parciales

En vez de usar *n* claves PRAC se podría usar una sola clave y una función no reversible de *m* bits a *m* bits para generar la lista de llaves PRAC requeridas. Cuando el agente es enviado inicialmente al servidor S_1 contiene la clave k_1 la cual es almacenada en el servidor original. Cuando el agente se prepara para ir del servidor S_{i+1} él obtiene $k_{i+1} = f(k_i)$ donde *f* es una función no reversible y entonces elimina k_i.

Al igual que antes un servidor malicioso S_k no podría falsificar los PRACs de los servidores anteriores, porque tendría que romper la suposición de la función irreversible o romper la función MAC.

Cuando el servidor original reciba los mensajes con la llave inicial y la función irreversible podrá volver a generar la llave para obtener los datos originales basta con saber el número del servidor que ha sido visitado.

Es importante señalar que esta técnica asume (como es obvio suponerlo) que todos los servidores no maliciosos no alteraran de ninguna manera ni las llaves ni las funciones que intervienes para su generación consecutiva. De esta forma se garantiza la obtención correcta de resultados hasta que se llegue con un servidor malicioso el cual sería bastante difícil de implementar.

14.2.5 Protección de datos mediante composición de matrices

Una forma simple de lograr la encriptación de los datos es mediante el uso de matrices, la desventaja de esta técnica es que se ve limitada a un restringido número de aplicaciones y en donde la mayor parte de los recorridos son sólo de una visita.

Consiste en poner los datos a evaluar en una matriz, (por ejemplo un mapa linear) estos datos sufrirán un cambio uniforme como podría ser un *zoom* para aumentar o disminuir el tamaño del mapa. Una vez colocados los datos en la matriz A se genera aleatoriamente una matriz S que se pueda invertir y se calcula $B=SA$. Entonces se envía B a servidor remoto con el agente, en dicho servidor se computa $y=Bx$ y el agente regresa con y a su servidor de origen. El servidor local obtiene $S^{-1}y$ y obtiene el resultado de Ax sin darle oportunidad al servidor remoto de conocer A.

Bajo este esquema es imposible que el servidor malicioso pueda espiar o hacer mal uso de la información porque no hay forma de obtener los valores reales de los datos.

14.2.6 Protección de datos mediante funciones homomórficas[79]

Finalmente presento una técnica que brinda una solución total a la protección de datos desde el punto de vista que un servidor malicioso no puede acceder de ninguna forma a los valores ya que estos son evaluados de forma encriptada. Desgraciadamente la implementación real de esta técnica en un sistema de agentes móviles sería muy costosa ya que implicaría posibles modificaciones en los interpretes al momento de evaluar los datos si se desea transparencia para el programador. Debido a esto dicho esquema de protección no será implementado en la primera versión de SAHARA.

Este esquema se basa en las propiedades de las funciones homomórficas que sirven para computar datos encriptados de forma no interactiva. Si una función E es homomórfica entonces $E(x+y)$ y $E(xy)$ son fáciles de obtener de $E(x)$ y $E(y)$.

Las posibilidades de la computación de datos y funciones encriptadas están en función de las propiedades homomórficas de la función y éstas pueden ser:

Homomórfica en la adición: si existe un algoritmo MAS que compute $E(x+y)$ de $E(x)$ y $E(y)$ y que no revele x y y.

---

[79] El termino homomórfico se ha usado en la jerga matemática como traducción de la palabra inglesa *homomorphic*, no obstante dicho termino no existe en castellano.

Homomórfica en la multiplicación: si existe un algoritmo MULT que compute $E(xy)$ de $E(x)$ y $E(y)$ y que no revele x y y.
Homomórfica mezclada en la multiplicación: si existe un algoritmo MULT-MEZ que compute $E(xy)$ de $E(x)$ y y, y que no revele x.
Homomórfica algebraicamente: si es homomórfica en la suma y en la multiplicación.

Para aclarar el concepto de homomorfismo daré un ejemplo. Supongamos que se quiere sumar x y y. Se define como función de encriptación $E(x)=e^x$ por lo que al aplicarlo a nuestros valores tendremos e^x y e^y respectivamente, en este momento nosotros dejamos de conocer el valor real de x y de y. Enseguida se define el algoritmo MAS que se encarga de multiplicar los valores $e^x * e^y$ que es igual a $e^{x+y} = E(x+y)$ de donde se puede obtener fácilmente $x+y$ al aplicar la función inversa *Ln*. Debe observarse que cualquier dato puede ser procesado sin conocer en ningún momento su valor, esto es posible porque la función $E(x)= e^x$ es homomórfica con respecto a la suma.

De acuerdo a lo descrito en el párrafo anterior un sistema de agentes puede definir una función algebraicamente homomórfica no reversible fácilmente[80] (como en el caso de e^x) que aplique a todos los datos que utilicen los agentes antes de salir del servidor logrando un encriptación de la información. Enseguida el agente define los algoritmos MAS y MULT de acuerdo a la función definida que le permita realizar sumas y multiplicaciones durante su recorrido.

El código del agente hará llamadas a dichos métodos para realizar sumas y multiplicaciones sin desencriptar los datos impidiendo con ellos el robo o mal uso de información por parte de un servidor malicioso. Al regresar podrá aplicar la función inversa de la función definida con gran costo computacional (debido a la dificultad supuesta para invertir la función) para poder obtener los resultados reales.

La mayor dificultad de esta técnica radica en la dificultad de encontrar la función apropiada que satisfaga todas las necesidades requeridas así como la dificultad de que el servidor puede aplicar la función a todos los datos definidos por un usuario programador de agentes móviles

14.3 ¿QUÉ SE LOGRA CON LA PROTECCIÓN DEL CÓDIGO Y DATOS?

Al proteger el código de los agentes mediante firmas digitales los usuarios podrán tener la certeza de que ningún servidor malicioso podrá modificar su comportamiento que vaya a afectar en las acciones que realicen en el resto de servidores que visiten durante su itinerario.

La protección de los datos que se ha convertido en la más ardua labor de los investigadores de agentes móviles se logra parcialmente ya que SAHARA utilizará códigos de autenticación de resultados parciales con lo que sólo podemos garantizar que los datos obtenidos hasta antes de llegar a un servidor malicioso serán confiables. En la

[80] Con no reversible fácilmente me refiero a que dicha función requiera de un largo proceso para obtener su inversa con lo que un servidor malicioso se vea imposibilitado de descifrarla.

mayor parte de los casos en donde el agente se ejecute en redes seguras o no se tope con servidores maliciosos los resultados del agente serán 100% confiables.

Merece la peña señalar que si bien es difícil proteger a los agentes también es muy difícil alterar su código o sus datos sin dañar su estructura básica para que este pueda seguir funcionando. La protección de datos es el único punto parcialmente vulnerable que tiene la arquitectura de SAHARA. La protección total de datos es el punto principal de mi trabajo futuro.

Capítulo XV
SAHARA: AUTOCONTROL DE RECURSOS Y ALCANCES DE LA ARQUITECTURA.

Quizá una de las características más importantes que posee SAHARA es la capacidad de verificar automáticamente su estado y controlar los recursos que le queden disponibles.

El sistema podrá ser programado para que periódicamente revise los recursos que quedan disponibles en sistema, y una vez que se sobrepasen los límites mínimos establecidos, el sistema inteligentemente podrá alterar los permisos otorgados con el fin de cuidar los recursos que le quedan y de esta forma mantener la estabilidad del sistema.

La arquitectura de SAHARA ofrece una solución total para casi todos los problemas de seguridad que enfrenta un sistema de agentes. No obstante, presenta una solución parcial para proteger el estado de datos de los agentes en contra de ataques de servidores maliciosos.

La idea principal para proteger los datos estará basada en el uso de diversas firmas para cada agente de tal forma que el servidor que reenvíe un agente asuma cierta responsabilidad de no haber modificado sus datos certificándolo con su propia firma.

15.1 AUTOCONTROL DE RECURSOS

La arquitectura estará dotada de inteligencia para poder frenar el consumo de recursos cuando estos hayan descendido de los límites establecidos. De igual forma, será capaz de restablecer los permisos que haya restringido temporalmente con afán de mantener la estabilidad del sistema cuando el servidor se haya recuperado de los recursos agotados.

Para asegurar la integridad del sistema SAHARA realizará constantemente una verificación de los recursos con los que cuenta el sistema y los comparará con los

límites mínimos establecidos para cada recurso (espacio en disco, RAM, conexiones de red, etc) los cuales serán determinados por cada administrador y se encontraran en el fichero *SAHARA/config/quotaMin.cfg*. Por defecto el sistema instalará este fichero sugiriendo los valores recomendados, siendo posible que el administrador los modifique de acuerdo a sus necesidades.

Si al hacer una verificación se detecta que algún recurso a descendido de los limites establecidos en *quotaMin.cfg*, el sistema inmediatamente ejecutará las tareas necesarias para evitar que se siga consumiendo ese recurso, comúnmente alterando el fichero que determina las políticas de seguridad del sistema, no sin antes hacer una copia de dicho fichero que restaurará cuando el servidor vuelta a tener la disponibilidad de ese recurso.

Una vez que entra en vigor la nueva política restrictiva del sistema la verificación de todos los recursos continua, cuando se detecta que el sistema se ha recuperado del recurso agotado entonces restaurará la copia del fichero de políticas de seguridad para volver a instalar los privilegios inicialmente otorgados.

Para saber cuando el servidor de agentes debe volver a restaurar la copia de seguridad se comparará constantemente el valor obtenido de la verificación con los especificados en el fichero *SAHARA/config/quotaMax.cfg* que define las cantidades que se consideran apropiadas para que el sistema pueda otorgar una vez más cualquier tipo de recursos.

Al igual que con el otro fichero, *quotaMax.cfg* es instalado por el sistema con los valores recomendados, sin embargo, el administrador del sistema será capaz de modificarlos.

La idea fundamental de este esquema de autocontrol consiste en asegurar que los agentes que se encuentran en ejecución puedan hacerlo hasta la consecución total de los resultados deseados. Es decir, se trata de prever el agotamiento total de los recursos del sistema, ya que de agotarse inevitablemente se tendría que negar dicho recurso o un servicio lo que conlleva a que los agentes en el sistema no podrían terminarse de ejecutar satisfactoriamente.

Al modificar los permisos del fichero de políticas de seguridad se evita que los nuevos agentes que lleguen al sistema no puedan consumir más ese recurso, para permitir a los que ya se les había concedido previamente que terminen de ejecutarse.

15.1.1 Autocontrol del espacio en Disco

Para restringir el consumo del espacio en disco se edita el fichero de políticas de seguridad del sistema llamado *system.policy*, en él se modifican todas la entradas *grant* que contengan permisos de escritura en el sistema de ficheros, es decir las entradas:

```
FilePermission p = new FilePermission(“MiArchivo”, “read,write”);
```

Serán cambiadas por entradas:

```
FilePermission p = new FilePermission(“MiArchivo”, “read”);
```

Con esto se logra que todos los nuevos agentes que lleguen al sistema no puedan consumir más este recurso, dejando las cantidades aún disponibles para que los agentes que ya se encontraban en el sistema terminen de ejecutarse.

Antes de realizar dicho cambio en el fichero *system.policy* se hará una copia que conserve los privilegios originales otorgados por el administrador para que se restaure en el momento que en que haya migrado un número suficiente de agentes que le haya permitido al sistema recuperar parte de los recursos consumidos.

15.1.2 Autocontrol de la memoria RAM

Cuando se detecte escasez de memoria RAM, el sistema no permitirá que los agentes activos consuman mas recursos. Por ejemplo, evitará que puedan duplicarse, de igual forma no permitirá la creación de agentes hijos.

El sistema, con afán de mantener su estabilidad ignorará las asignaciones previamente concedidas a un agente en donde se especifica el número de hijos o copias que éste puede tener. Es decir, si el sistema desciende del límite señalado en *quotaMin.cfg*, no permitirá que un agente pueda crear nuevos agentes a pesar de que éste no haya llegado al límite que le fue impuesto en su asignación.

Adicionalmente el servidor no recibirá más agentes de ningún sistema remoto para evitar al máximo el consumo de memoria. El sistema en cuestión permanecerá en este estado hasta que al migrar algunos agentes se recuperen parte de los recursos que habían sido consumidos y el límite establecido en *quotaMax.cfg* sea sobrepasado.

Cuando el sistema vuelva a ser estable, entrará en vigor la política de seguridad que había sido establecida originalmente en ese servidor, es decir, serán consideradas las asignaciones inicialmente establecidas.

15.1.3 Autocontrol de las conexiones a la red

Cuando se detecte sobresaturación de la red, será necesario reducir el número de conexiones para disminuir la cantidad de tráfico. Para ello serán restringidas las acciones "*accept*" y "*connect*", que son la que generan tráfico fuera del servidor.

Al igual que para controlar el espacio de disco se editará el fichero de políticas de seguridad del sistema llamado *system.policy*, en él se eliminarán todas las líneas en las entradas *grant* que contengan permisos de conexión y recepción de *sockets* a través de cualquier puerto del ordenador, es decir, cualquier entrada que contenga permisos "*accept*" y "*connect*".

```
p = new SocketPermission("*.uniovi.es:-1023","accept");
p = new SocketPermission("*.uniovi.es:1024-","connect");
```

De esta manera los nuevos agentes que lleguen al sistema no podrán hacer uso de la red y los recursos disponibles quedarán a disposición de los agentes que ya se encontraban en el sistema.

Antes de modificar el fichero *system.policy* se hará una copia que conserve los privilegios originales otorgados por el administrador para que se restaure en el momento que en que haya migrado un número suficiente de agentes que disminuya el tráfico de la red.

Para el caso de las conexiones a red el administrador podrá fijar dos valores en *quotaMax.cfg,* en cuanto el sistema recupere recursos y sobrepase el primer valor podría restringir sólo los permisos de "*connect*" para dar la posibilidad de ejecutar aquellas aplicaciones que solo requieran recibir información, no obstante al no permitir abrir nuevas conexiones se logra menos saturación de la red.

Si sólo hay un valor en *quotaMax.cfg*, el sistema esperará a que se alcance el valor establecido para restablecer el fichero de políticas de seguridad original y con ello los privilegios originales que el administrador había otorgado.

15.2 ALCANCES Y LIMITACIONES DE LA ARQUITECTURA

SAHARA es una arquitectura ambiciosa que da solución integral a la mayor parte de los problemas de seguridad que un sistema de agentes puede tener, no obstante, para que dicha arquitectura pueda ser implantada en un sistema de agentes, éste debe reunir ciertos requerimientos que son necesarios para que el sistema de seguridad pueda ser implantado.

Como no se ha definido un estándar en los requerimientos de los sistemas de agentes móviles, es probable que algunos sistemas carezcan de ellos y que todas las bondades de SAHARA no puedan ser implantadas. Estos requerimientos representan en cierta forma una restricción y por ello debemos mencionar los alcances actuales de esta arquitectura así como las limitaciones de implementación.

15.2.1 Alcances de SAHARA

La arquitectura deja algunos aspectos de seguridad que serán mejorados en versiones futuras, por ejemplo, el dar una solución completa al problema de servidores maliciosos. Presentaré a continuación los alcances de SAHARA:

- *Solución parcial al problema del servidor malicioso:* la arquitectura sólo ofrece protección del código de los agentes mediante firmas digitales y proporciona una serie de métodos que permiten proteger parcialmente los datos, como los PRAC. También introduce la idea de la firma de los datos por parte de los servidores remotos para adjudicar responsabilidad. No obstante no ofrece una solución total a dicho problema.

- *Carece de un esquema de protección de mensajes:* SAHARA no proporciona ningún esquema de protección para la encriptación, validación y recepción de mensajes. Si algún sistema desea incorporara la arquitectura de SAHARA y cuenta con soporte para el envío de mensajes síncronos y asíncronos, éstos no serán protegidos por ningún medio y podrían ser alterados o falsificados por otras entidades malévolas.
- *No cuenta con un mecanismo para la distribución automática de certificados de claves públicas:* para la validación de agentes los sistemas remotos requieren la incorporación de un certificado de clave pública de la autoridad que desee enviar agentes a dicho servidor remoto. SAHARA no incluye un mecanismo para distribuir estos certificados automáticamente, cada vez que se desea incorporar una autoridad a una red de servidores, es necesario que cada uno de éstos utilice la opción *importar certificado* para que esta autoridad pueda empezar a enviar agentes a estos servidores.
- *Carece de medios para una gestión segura y eficiente de clases repetidas:* SAHARA ofrece mecanismos seguros y eficientes para la protección y envío de clases pero no proporciona ningún medio para tratar las clases repetidas que sean de agentes distintos o bien las clases repetidas que sean de una misma autoridad pero de versiones diferentes. SAHARA actualmente protege las clases adjuntas al agente y estás son cargadas y gestionadas desde dentro del propio fichero *Jar* para evitar los problemas anteriores.
- *Navegación libre restringida:* SAHARA viene a ser un poco restrictiva ya que el administrador tiene que dar de alta las autoridades remotas que podrán tener acceso al sistema con lo que se limita en cierta medida la libre navegación de los agentes en Internet. No obstante esto puede interpretarse como un mal no deseado de la seguridad, ya que el administrador puede otorgar privilegios a autoridades no autentificadas (con las que podría entrar cualquier agente) pero corre el riesgo de sufrir ataques por grupos no identificados.

Adicionalmente a las carencias que marcan los alcances actuales de la arquitectura de SAHARA, la implementación total de la arquitectura tiene algunas dependencias del lenguaje sobre el que se implemente, las cuales se describen a continuación.

15.2.2 Limitaciones de implementación de SAHARA

SAHARA es dependiente de las capacidades del lenguaje en que se diseñe el sistema de agentes sobre el cual será implantada la arquitectura. Es decir, para que puedan implementarse todas las verificaciones de seguridad y se aprovechen todas las características de la arquitectura en necesario que el lenguaje de soporte a muchas tareas que se requieren. A continuación se describen las más importantes para la implementación de la arquitectura.

- *Multiplataforma:* es deseable que el lenguaje sobre el que se desarrolla pueda ser ejecutado en varias plataformas para que los agentes puedan navegar libremente y se satisfaga el requerimiento de portabilidad.
- *Orientación a objetos:* es necesario que el lenguaje soporte la orientación a objetos para que todo el paquete de clases que define la funcionalidad del sistema de seguridad pueda ser implantado y gestionado como lo señala SAHARA.

- *Multihilo:* el lenguaje deberá soportar la programación multihilo o multiproceso para que cada agente pueda ser representado como un hilo o proceso de ejecución independiente y a su vez se permita la ejecución de varios de ellos en forma concurrente.
- *Posibilidad de verificación del consumo de recursos:* para la gestión apropiada de las asignaciones impuestas a cada agente es necesario que el lenguaje permita contabilizar el consumo individual por cada hilo de ejecución o agente de Disco duro, RAM, conexiones de red, para de esta manera poder restringir el consumo de discos, el consumo de memoria RAM y del resto de los recursos del sistema[81].
- *Extensibilidad:* es deseable que el lenguaje soporte la integración de componentes para la incorporación de módulos ya programados. De esta manera se puede implementar el sistema en forma modular en donde los módulos se invoquen unos a otros y se favorezca la reutilización. De esta manera podría crearse un protocolo de transferencia de agentes independiente de la plataforma.
- *Soporte para seguridad:* es deseable que el lenguaje cuente con algunas interfaces de seguridad creadas como aquellas que permitan crear y validar firmas digitales, encriptar información, etc. De esta manera se facilitaría las tareas de desarrollo de la arquitectura.

[81] Java2 versión 1.2.1 cuenta con muchas interfaces para la verificación del consumo de recursos que serán usadas a la hora de implementar este prototipo.

PARTE IV .- EL PROTOTIPO MILENIO

Capítulo XVI
EL PROTOTIPO MILENIO: UN SISTEMA DE AGENTES MOVILES CON SEGURIDAD INTEGRAL

Para poder implementar la arquitectura de SAHARA es necesario tener un sistema de agentes disponible sobre el cual se pueda implantar. Por razones de diseño se requiere tener el código fuente del sistema de agentes, de igual forma es necesario que dicho sistema haya sido desarrollado en el mismo lenguaje y versión que el que se desea usar para el desarrollo de la arquitectura de seguridad SAHARA.

Como no es posible contar con el código fuente completo de un sistema de agentes móviles comercial, para implementar esta tesis he decidido desarrollar no sólo la arquitectura de seguridad sino el sistema de agentes móviles que la soporte. A dicho sistema le llamaré MILENIO.

Dentro de este capítulo describiré la filosofía de implementación del primer prototipo de la arquitectura de seguridad SAHARA así como la del primer sistema de agentes móviles que la contiene, es decir, MILENIO. Mostraré los módulos y jerarquías de clases más importantes y los detalles fundamentales de su implementación.

El prototipo de la arquitectura de seguridad SAHARA así como el prototipo de MILENIO fueron programados como dos proyectos de fin de carrera de la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo. Ambos proyectos al ser programados se basaron en las especificaciones detalladas de esta tesis. Para profundizar en los detalles de la implementación se pueden consultar [HIDA99] y [SERAFIN].

MILENIO como sistema de agentes prototipo contará con los elementos básicos para que un sistema de agentes pueda funcionar, carecerá de entornos de desarrollo sofisticados con el que ya cuentan algunos sistemas de agentes móviles así como utilidades que son en ocasiones innecesarias como la posibilidad de envío y recepción de mensajes por parte de los agentes.

No obstante, el sistema será desarrollado concienzudamente para que mejore la eficiencia de los sistemas existentes en el mercado. Por ejemplo, se mejorará la técnica

de envío y recepción de las clases que requiere un agente para poder ser instanciado en un servidor remoto.

No obstante, MILENIO contará con la arquitectura de seguridad integral SAHARA[82] lo que le convertirá en un sistema de agentes móviles único. Al implementarse la arquitectura de SAHARA será el sistema de agentes más seguro actualmente. Se trata pues, de desarrollar una implementación que permita comprobar en la práctica el funcionamiento de las ideas descritas en el diseño de SAHARA.

16.1 DEFINICIÓN DEL LENGUAJE

Para implementar MILENIO y la arquitectura de seguridad SAHARA se debe seleccionar el lenguaje que más facilite la programación de las tareas que un sistema de agentes debe desempeñar, y que facilite satisfacer las necesidades que el sistema debe cubrir.

De acuerdo a los requerimientos de un sistema de agentes descritos en el capítulo X el lenguaje en el que se programe el sistema deber ser:

- Un interprete portable a la mayor cantidad posible de plataformas para que los agentes móviles puedan navegar en todo Internet.
- Orientado a objetos y fácilmente escalable mediante la incorporación de componentes para facilitar un diseño modular.
- Versátil, al evaluar y restringir la cantidad de recursos consumidos.
- Seguro, permitiendo políticas de seguridad en donde se puedan conceder permisos de forma específica.
- Capaz de serializar cualquier tipo de objeto para poder enviar el estado de los agentes.
- Multihilo y multitarea para tener múltiples agentes en ejecución.
- Capaz de invocar métodos de objetos remotos para establecer protocolos de comunicación y enviar clases.
- Etc.

Los requerimientos anteriores han sido mencionados en orden de importancia de acuerdo a las necesidades de implementación de un sistema de agentes y de la arquitectura de seguridad SAHARA. Actualmente el lenguaje de programación que permite implementar una mayor parte de estos requerimientos es el lenguaje de programación de Java.

Los sistemas de agentes móviles que actualmente tienen más auge en el mercado están implementados en Java, la mayor parte de ellos son sistemas comerciales que constantemente están evolucionando para satisfacer las nuevas demandas de los usuarios, entre ellos se pueden citar a los Aglets [LANG96], Concordia[WALS98], Voyager[OBJE97] y JumpeansBeans [ASTR].

[82] El demostrar el funcionamiento de esta arquitectura es el objetivo fundamental de este prototipo ya que representa una arquitectura de seguridad robusta, única en su genero para los sistemas de agentes móviles basados en Java existentes en el mercado.

16.1.1 Justificación del lenguaje Java 2 SDK (JDK 1.2)

Java hoy en día es un lenguaje de desarrollo maduro y ampliamente difundido, su interprete puede ser ejecutado en todas las plataformas más importantes del mercado. Es orientado a objetos y soporta la incorporación de paquetes.

Adicionalmente la nueva versión del JDK 1.2 llamada Java 2 SDK[83] incorpora un nuevo esquema de seguridad que facilita asignación de permisos individuales a las clases u objetos provenientes de la red.

Una carencia de Java2 SDK es que no permite contabilizar los recursos del sistema (RAM, HD, etc.) consumidos en forma individual por un hilo o bien por una clase en ejecución. No obstante satisface la mayor parte de los requerimientos señalados anteriormente por lo que resulta ideal para la desarrollo del prototipo de MILENIO y sobre todo para la implementación de la arquitectura de SAHARA.

16.2 DESCRIPCIÓN GENERAL DEL PROTOTIPO MILENIO

El marco de trabajo de MILENIO consta de tres capas principales. En la capa superior que corresponde a la capa de aplicación se encuentran todos los agentes definidos por el usuario. La siguiente capa corresponde a la API y a la implementación del sistema de agentes móviles MILENIO que se encarga de ejecutar y administrar a todos los agentes activos en el sistema.

Figura 16.2.1 – Marco de trabajo de MILENIO

[83] Las antiguas versiones del JKD 1.1.x no soportaban un esquema de seguridad versátil en donde se pudieran conceder privilegios individuales.

Estas dos capas subyacen sobre el protocolo de comunicación a través del cual se envían los agentes, que para el caso de MILENIO se requería un protocolo ampliamente difundido y que permitiera la transmisión de agentes a cualquier parte del mundo por lo que indudablemente se escogió el protocolo de Internet, es decir, TCP/IP. Esto puede apreciarse con detalle en la figura 16.2.1.

La capa correspondiente al sistema MILENIO que se encarga de la gestión apropiada y ejecución de agentes consta de tres módulos: el núcleo, componentes de administración del sistema y de la arquitectura de seguridad SAHARA.

El sistema de los aglets [LANG98b] posee módulos similares salvo que su protocolo de transferencia de agentes es independiente de la plataforma y se monta sobre http, sin embargo aunque tiene un componente de seguridad, este muy lejos de proporcionar una seguridad integral y la versatilidad que proporciona SAHARA. El resto de los sistemas de agentes basados en Java tampoco proporcionan un módulo o un sistema de seguridad confiable. Por su parte ARA [KAISb] presenta una división diferente. Como soporta interpretes diferentes para la creación de agentes existe una estrecha relación entre cada interprete de la capa de aplicación y el núcleo.

Volviendo a MILENIO cada módulo a su vez consta de varios componentes o clases que se encargan de realizar una tarea particular, la división de tareas del sistema llevada a cabo por sus respectivos componentes se puede apreciar en la siguiente gráfica.

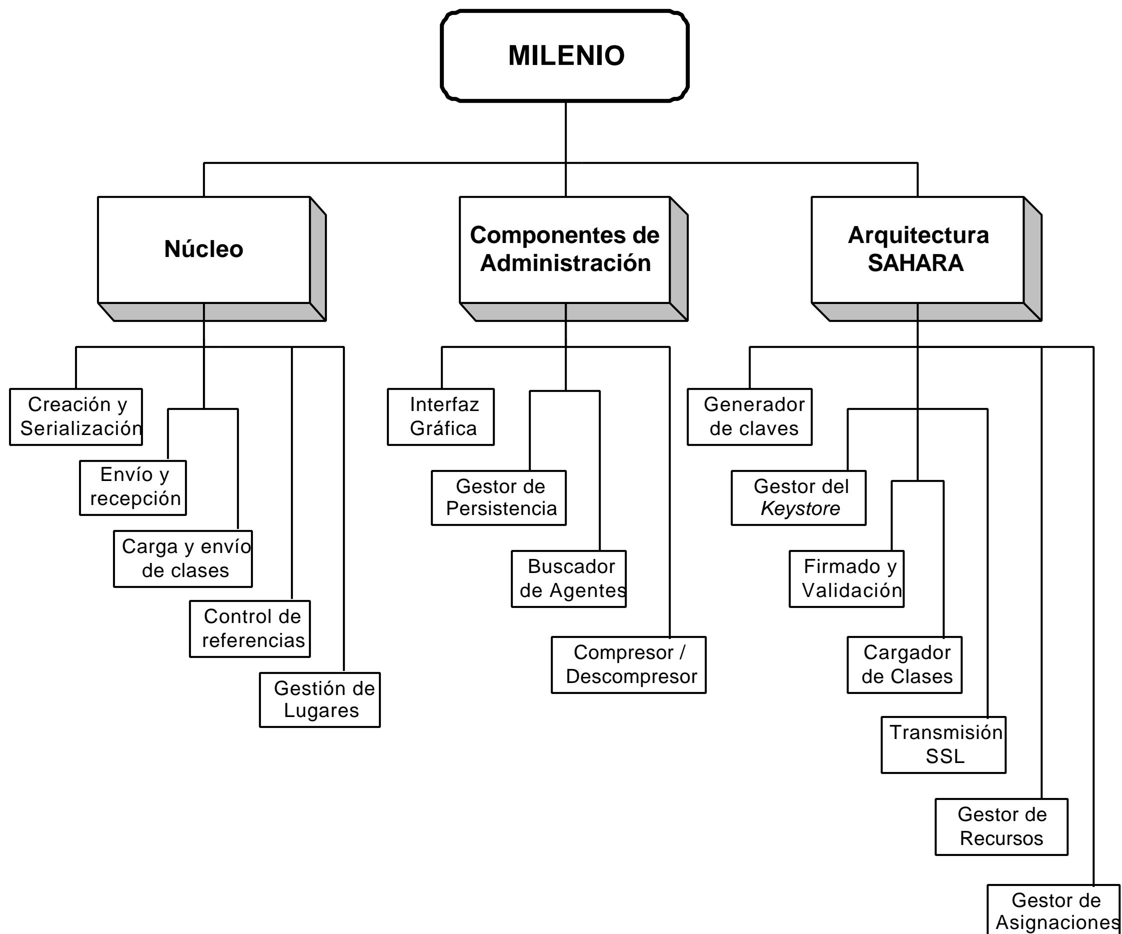


Figura 16.2.2 – Componentes de MILENIO

A continuación se describe la función general de cada módulo y de algunos componentes del sistema.

16.2.1 El núcleo

El núcleo es la parte fundamental del sistema de agentes ya que se encarga de realizar las tareas básicas y de proporcionar los servicios necesarios para la ejecución de agentes, por ejemplo creación, duplicación, envío, recuperación, desactivación, activación, etc. De igual forma se encarga de reiniciar a los agentes remotos provenientes de diversos sitios. Las funciones principales del núcleo son:

- *Definición, inicialización, serialización y deserialización de agentes:* se encarga de crear e inicializar agentes así como de serializarlos antes de que sean enviados.
- *Envío y recepción de agentes:* envía a los agentes que lo han solicitado y recibe a los agentes autorizados.
- *Gestor de clases de agentes locales:* carga las clases de los agentes locales y envía las clases requeridas por servidores remotos para poder ejecutar agentes que han migrado.
- *Control de las referencias de agentes y recolección de basura de los mismos:* contabiliza los agentes activos en el sistema y se encarga de la recolección de basura cuando éstos son eliminados.
- *Administración de lugares y regiones:* lleva el control de los lugares activos en donde se ejecutan agentes dentro de un sistema de agentes así como de las regiones a las que pertenece el sistema.
- *Sincronización de componentes:* se encarga de invocar apropiadamente todos los componentes de seguridad para garantizar una integridad de todo el sistema.

16.2.2 Componentes de administración del sistema

Estos componentes son usados para personalizar algunas características del sistema o bien para facilitar algunas de sus tareas y mejorar su desempeño. Los componentes son escalables e independientes lo que facilita la reincorporación de alguna nueva versión mejorada de alguno de ellos. No podrán ser accedidos por los programadores de agentes sino sólo por el administrador o bien por el núcleo del sistema. Entre ellos se encuentran:

- *Gestor de persistencia:* se encarga de almacenar y de recuperar a los agentes que han sido previamente desactivados. El gestor de persistencia almacena el *bytecode* y el estado de los agentes desactivados en un almacenamiento secundario.
- *Compresor/descompresor:* todos los agentes para aumentar su rapidez de transmisión viajarán comprimidos. Este módulo se encargará de comprimir la clase y el estado del agente en un fichero JAR antes de ser enviado y de descomprimirlo una vez que llega a su destino.

- *Interfaz gráfica:* permite al administrador personalizar y gestionar del sistema de una forma más amigable, este componente es invocado por el núcleo del sistema e interactúa indefinidamente con él.
- *Buscador y enrutador de agentes*[84]*:* se encarga de buscar un agente en un servidor local o remoto. De igual forma este módulo se encargará de proporcionar al agente el tipo de información que contiene el servidor local para que el agente pueda decidir si ésta es la información que necesita o bien para migrar a otro sitio en donde puede encontrar la información deseada. El enrutador sugerirá direcciones a donde pueda migrar para encontrar la información requerida[85].
- *Gestor de mensajes*[86]*:* Se encargará de la recepción y entrega de los mensajes que le sean enviados al agente desde destinos remotos así como de enviar los mensajes del agente a otros servidores.

16.2.3 Arquitectura de seguridad SAHARA

Este es el módulo fundamental de MILENIO, ya que este sistema de agentes se caracterizará por su eficiente seguridad así como de la versatilidad que ofrece para personalizar el entorno de seguridad de acuerdo a las necesidades particulares de cada sistema. El sistema cumple con todos los objetivos señalados en la arquitectura de seguridad que implementa.

La arquitectura de SAHARA será implementada en su totalidad[87] tal como fue descrita en los capítulos del XI al XV por lo que se puede afirmar que este módulo se encarga de garantizar la seguridad total del sistema de agentes y sus recursos así como de la integridad de los agentes móviles[88]. El módulo de seguridad consta de las siguientes partes:

- *Generador de claves:* se encarga de generar las claves públicas y privadas así como los certificados para todos las autoridades que se den de alta en el sistema. De igual forma le proporcionará a los agentes los pares de claves requeridas si éstos desean hacer un itinerario protegiendo sus datos mediante códigos de autenticación de resultados parciales.
- *Gestor del Keystore:* almacena las claves públicas y privadas para cada autoridad/usuario que se da de alta en el sistema así como los certificados de autenticación de los usuarios remotos. De igual forma recupera la clave privada solicitada en el proceso de firmado de un agente.
- *Gestor de firmado y validación de agentes:* realiza el proceso de firmar a todos los agentes que van a migrar con su respectivo estado de datos. La firma que utilizará para firmar el agente será la de la autoridad a quien pertenece y para firmar su

[84] En esta versión de MILENIO sólo se implementará la parte correspondiente al buscador.
[85] El servicio de enrutamiento es útil sobre todo en aplicaciones de comercio electrónico en donde un agente va en busca de un artículo o producto específico.
[86] Este módulo no será implementado en esta versión de MILENIO, no obstante el sistema estará preparado para su fácil incorporación.
[87] Excepto el consumo de disco y memoria RAM por agente ya que la plataforma de Java no ofrece actualmente medios para hacerlo. De igual forma en nuestro primer prototipo no implementamos los códigos de autenticación de resultados parciales (PRAC) para proteger los datos.
[88] Para conocer todos los mecanismos de protección que ofrece SAHARA refiérase al capítulo XI y XII.

estado de datos utilizará la firma del sistema de agentes en el que se encuentra. De igual forma este componente se encargará de verificar la validez de todas las firmas con que se hayan firmado todos los agentes remotos que pretendan entrar al sistema local.

- *Cargador de clases:* carga los agentes remotos a la vez que les asigna los permisos otorgados a la autoridad de dicho agente. Los permisos los asigna basándose en el fichero de políticas de seguridad del sistema y creando un dominio de protección para la autoridad propietaria de ese agente. Si el cargador de clases carga un agente de una autoridad que ya tiene creado un dominio de protección simplemente agrega ese agente al dominio de protección existente.
- *Capa de transmisión SSL:* establece una canal de comunicación privado para trasmitir los agentes móviles. Para establecer la conexión los servidores se encargan de autentificarse mutuamente para evitar que un servidor quiera hacerse pasar por otro. Este modulo garantiza que los agentes serán transferidos sin temor a ser interceptados por grupos terceros.
- *Gestor de recursos:* será el encargado de la contabilidad de los recursos consumidos así como de los recursos disponibles del sistema. Este componente será el encargado de modificar las políticas de seguridad del sistema para evitar que MILENIO llegue a estar inestable debido a la falta de recursos.
- *Gestor de asignaciones:* establece la asignación que le corresponde a cada agente a la hora de su creación basándose en el propietario o autoridad de dicho agente. Cuando se reciben agentes remotos se encarga de llevar la cuenta de los recursos consumidos por los agentes de dicha autoridad particular y de no permitir que éstos sobrepasen los límites que les han sido establecidos.

Es importante indicar que la arquitectura de seguridad SAHARA esta integrada con el sistema, es decir, todas las funciones para encriptar y firmar información se invocan internamente y no como el caso de *D'Agents* en donde para encriptar información invocan un programa independiente que usa PGP para encriptar información. El uso de librerías o programas externos aumentan más aún el tiempo de respuesta de la aplicación.

16.2.4 Diagrama general de clases

Una vez descrito cada uno de los módulos y los componentes principales de MILENIO, se puede hacer un primer acercamiento a un diagrama general de clases que muestra las principales clases del sistema. Aunque es imposible una separación total debido a una interdependencia, el sistema consta de dos grandes módulos: el núcleo y sistema de agentes (que incluye los componentes de administración) y la arquitectura de seguridad de SAHARA. Ambos son presentados a continuación.

16.2.4.1 Diagrama general de clases del Núcleo y Sistema de Agentes

Al arrancar el sistema lo primero que se ejecuta es la inicialización del núcleo <<Clase AgentSystem>>. Es posible pasarle parámetros en línea, de no ser así toma los valores por defecto y se definen sus dos estructuras de datos <<NameAgentSystem y

Properties>> (véase figura 16.2.4.1.1). Enseguida (aunque no lo muestra el diagrama de clases debido a que sólo se muestras las clases principales) se inicializa su interfaz gráfica que será el medio de interacción con los usuarios definidos en el sistema.

El núcleo del sistema deberá actuar como un monitor interrumpido y en "paralelo" de todos los servicios demandados por las entidades que pueden participar en la ejecución de una aplicación. Es decir, deberá atender las peticiones de los usuarios a través de la interfaz gráfica, al igual deberá satisfacer las demandas de los agentes que se encuentran en ejecución mediante *AgentServices* así como estar al tanto de las peticiones de envío hechas desde servidores de agentes remotos con *SSLReceiver* y de las verificaciones de seguridad que éstas implican. Una vez detectado el servicio demandado éste lo canalizará a la clase o gestor apropiado para continuar con su tarea de supervisor y proveedor de servicios.

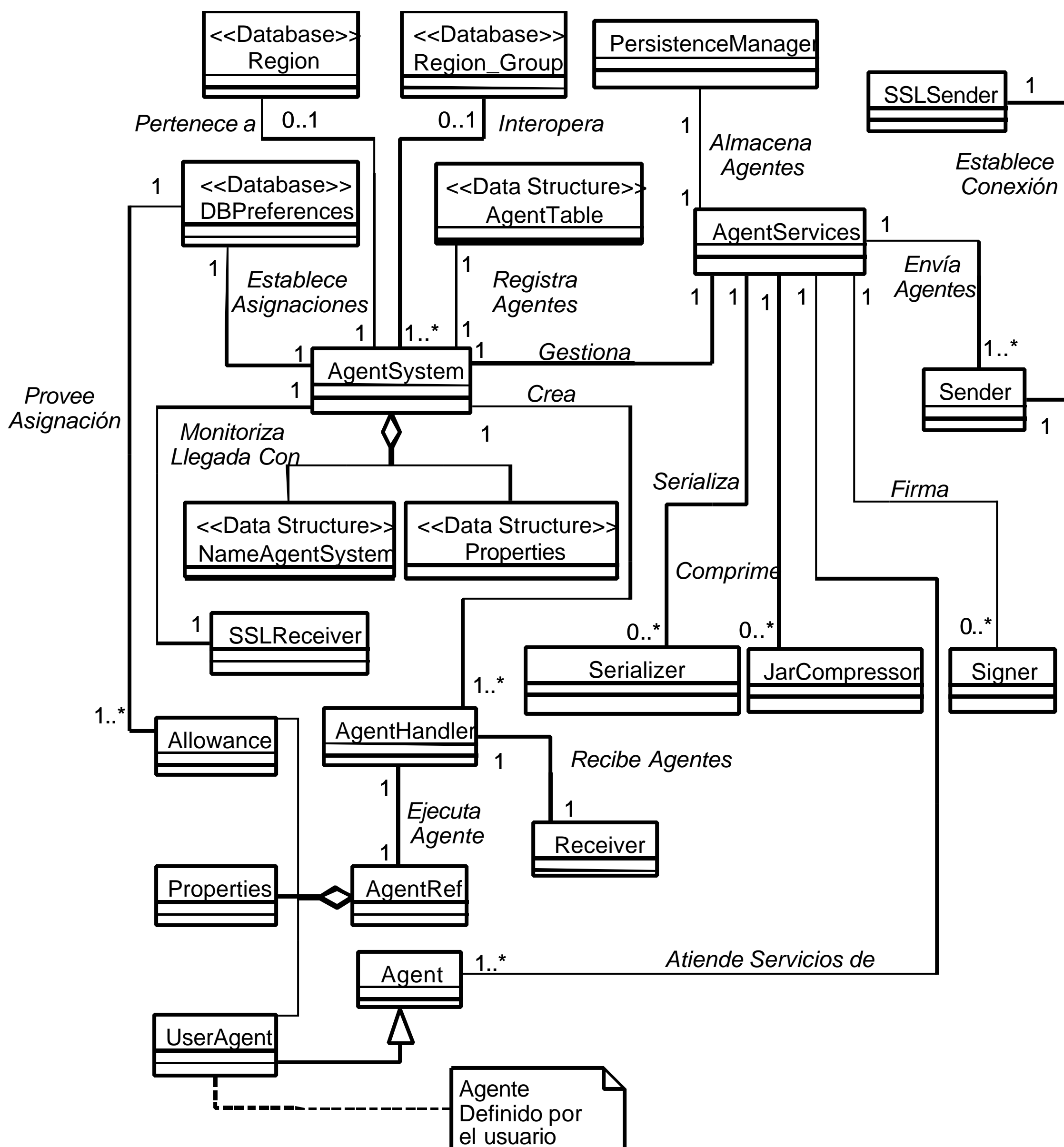


Figura 16.2.4.1.1 – Diagrama General de Clases del Núcleo y del Sistema de Agentes

Cuando el núcleo recibe una petición de creación ya sea de un usuario o de un agente, éste crea una instancia de la clase *AgentHandler* que se encarga de invocar el constructor *Oncreation()* del agente así como de solicitar y crear un objeto *AgentRef* que contiene todos los elementos de un agente, es decir, es el agente en sí ya que contiene el agente definido por el usuario[89] <<UserAgent>>, sus propiedades <<Properties>> y la asignación <<Allowance>> que es tomada de la base de datos *DBPreferences*. El *AgentHandler* crea un identificador único para el agente asociado con su *AgentRef* y se lo pasa al núcleo para que los de alta en la tabla de referencia de agentes activos en el sistema llamada *AgentTable*. Finalmente *AgentHandler* solicita un hilo de ejecución para el agente y ejecuta en él a *AgentRef*.

Cuando el núcleo recibe una solicitud de envío desde un sistema remoto actúa de forma muy parecida que cuando crea un agente, es decir, después de la autenticación del sistema remoto crea una instancia de *AgentHandler,* sin embargo antes de continuar con el proceso descrito anteriormente éste crea una instancia de la clase *Receiver* que se encarga de recibir al agente que es enviado desde un servidor remoto.

Por último cuando el núcleo recibe una petición de envió por parte de un agente este proporciona el servicio a través de su gestor de servicios llamado *AgentServices*. Para enviar al agente *AgentServices* invoca métodos de *Serializer* para serializar el estado de datos del agente, de *JarCompressor* para comprimir todos sus objetos y clases dentro de un fichero Jar y de *Signer* para firmar el agente con la firma de la autoridad a la que pertenece. Finalmente crea una instancia de la clase *Sender* que se encarga de contactar con el servidor remoto, de iniciar una conexión SSL mediante *SSLSender* y de hacer el envío. Para profundizar en los detalles de envío consulte el tema 16.3.6 protocolo de transferencia de agentes.

Resulta oportuno mencionar que el concepto de asignación usado en SAHARA es muy diferente al que se usa en *Ara*, en SAHARA la asignación se agrega al agente y viaja con él, representa la cantidad máxima de recursos que una autoridad desea que cada uno de sus agentes consuma, mientras que en *Ara* la asignación representa los permisos que se le han otorgado a ese agente a la hora de entrar en un lugar de un sistema.

En SAHARA el concepto de asignación tiene otro significado ya que también representa la cantidad de recursos que se le otorgan a una autoridad remota, a esta asignación se le restarán las asignaciones individuales de los agentes para determinar la cantidad de recursos disponibles de los que dispone una autoridad en un sistema remoto.

16.2.4.2 Diagrama general de clases de la Arquitectura de Seguridad

Cuando se invoca la ejecución del sistema el núcleo invoca los métodos de la clase *Configuration* para definir las propiedades y rutas de acceso del sistema. De igual forma se encarga de arrancar el gestor de seguridad de Java llamado *SecurityManager*.

Cada vez que el administrador del sistema da de alta una nueva autoridad el núcleo le pide al gestor del repositorio de llaves llamado *KeyStoreManager* que genere por medio

[89] Que hereda de la clase *Agent* todos los métodos que le dan movilidad y características de agente móvil.

del *keyGenerator* un par de llaves que serán asociadas a la nueva autoridad y guardadas en el repositorio de llaves llamado *KeyStore*. La clase *Signer* solicita al *KeyStoreManager* la clave requerida para firmar al agente *AgentRef* cuando éste vaya a migrar. Obsérvese la figura 16.2.4.2.1.

Cuando un agente remoto es recibido el núcleo se encarga de crear varias clases y de invocar a varios gestores para que reinicien el agente, verifiquen su firma y hagan la asignación de privilegios así como la asociación a un dominio de protección apropiado. De igual forma el núcleo auxiliado por el gestor de recursos realiza la supervisión del consumo de recursos.

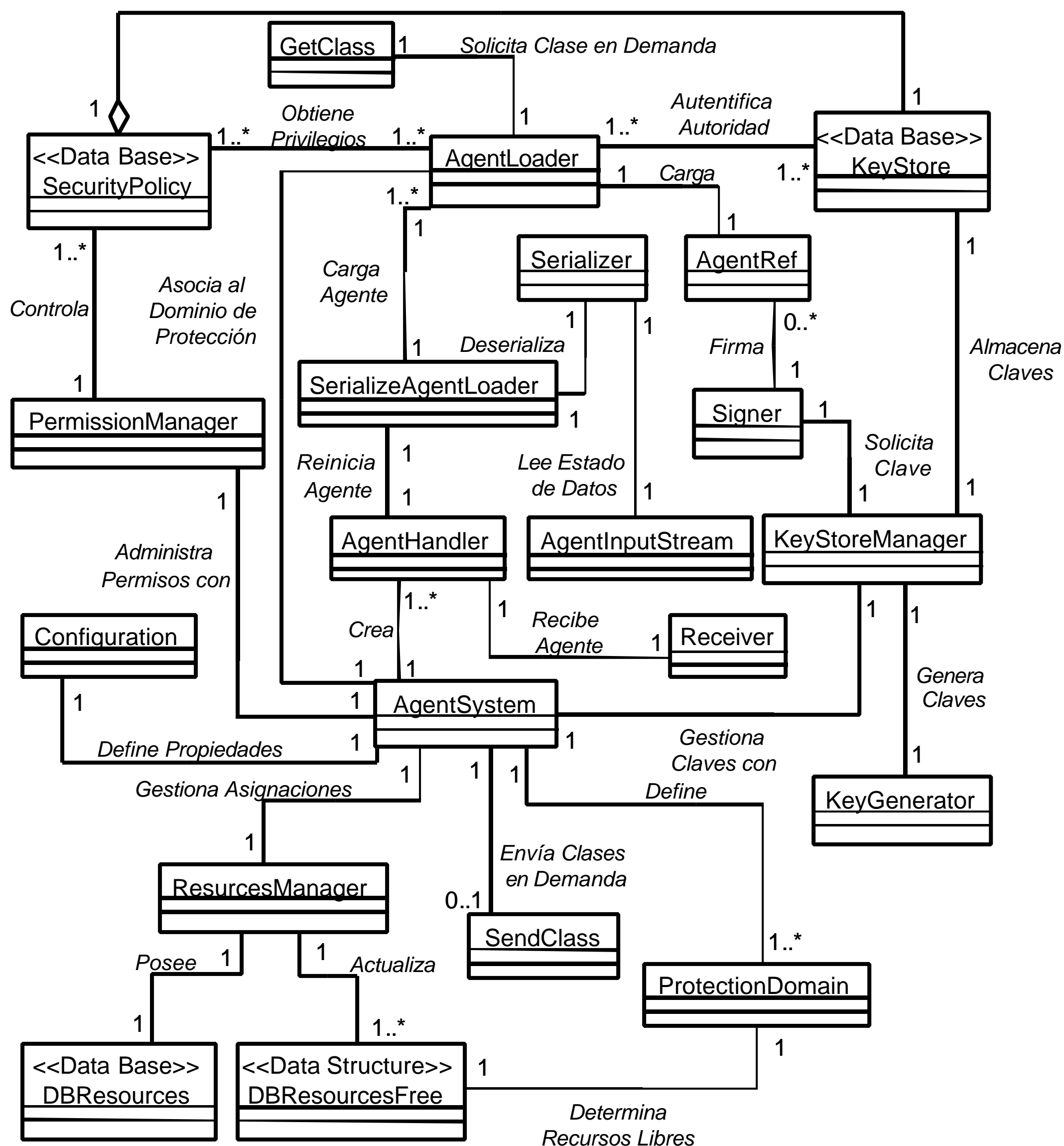


Figura 16.2.4.2.1 – Diagrama General de Clases de la Arquitectura de Seguridad de MILENIO

Cuando el núcleo es avisado por *SSLReceiver* (ver figura 16.2.4.1.1) de que un agente remoto será recibido, éste crea un *AgentHandler* (ver figura 16.2.4.2.1) que será el encargado de la recepción del agente, para ello crea una instancia de *Receiver* para que reciba el fichero Jar de agente. Después *AgentHandler* crea una instancia de

SerializeAgentLoader que se encargará de reiniciar al agente, pare ello crea una instancia de la clase *AgentLoader* que se encargara de la carga del agente *AgentRef* y sus clases, de su autenticación obteniendo la clave pública desde el *keyStore* y de la obtención de sus privilegios basándose en su autoridad desde la base de datos de políticas de seguridad <<Base de datos SecurityPolicy>>.

Posteriormente *SerializeAgentLoader* invoca a *Serializer* para que se encargue de deserializar al agente, éste a su vez llama a *AgentInputStream* para recuperar su estado de datos serializado. Finalmente tras todo este proceso el *AgentHandler* crea un hilo y reanuda la ejecución del agente.

El *AgentLoader* asocia al agente con el dominio de protección de su autoridad y se lo comunica al núcleo o en su defecto le pide la creación de un nuevo dominio de protección en caso de que no exista un dominio creado para esa autoridad. El núcleo entonces invoca al gestor de recursos <<Clase ResourcesManager>> para que actualice las asignaciones disponibles para ese dominio de protección <<DBResourcesFree>>. En caso de que el dominio de protección sea creado, para actualizar el objeto *DBResourcesFree* es necesario leer las asignaciones desde la base de datos *DBResources.*

Si durante su creación o su ejecución un agente requiere cargar una clase que no se encuentra dentro del fichero *Jar* en donde se encuentran comprimidas todas sus clases, entonces el *AgentLoader* crea un objeto *GetClass* que se encarga de solicitar en demanda la clase requerida. Para ello, *GetClass* establece contacto con el servidor de origen del agente quien crea un objeto *SendClass*, posteriormente abre una conexión SSL con *SendClass* quien se encarga de enviarle la clase solicitada. Este proceso se repite cada vez que una nueva clase inexistente sea solicitada.

Adicionalmente el núcleo encarga al gestor de recursos la supervisión de los recursos disponibles con que cuenta el sistema, en caso de que se estén agotando éste avisará al núcleo para que éste de la orden de modificar en tiempo de ejecución la política de seguridad establecida y mantener la integridad del sistema.

El gestor de permisos <<PermissionManager>> se encarga de modificar la base de datos de políticas de seguridad para asignar, cambiar o eliminar un privilegio de una autoridad, y de atender las peticiones hechas al núcleo cuando un agente quiera añadir un privilegio de acceso a un fichero en tiempo de ejecución. De igual forma el gestor de permisos modificará la política de seguridad en tiempo de ejecución cuando reciba la orden del núcleo quien habrá recibido una señal de alarma del gestor de recursos.

El diseño modular en clases y paquetes del sistema de seguridad favorece su fácil incorporación a sistemas de agentes móviles basados en Java, sólo bastaría con agregar los paquetes al sistema e incluir en el sitio apropiado las llamadas a las funciones de seguridad dentro del núcleo del sistema. Lo que quiero especificar es que el grado de dependencia de la arquitectura de seguridad con el núcleo es mínimo.

Capítulo XVII
MILENIO: DESCRIPCIÓN DETALLADA DE LA IMPLEMENTACIÓN DEL NÚCLEO

Ahora que se ha descrito el comportamiento general del sistema es el momento de precisar algunos aspectos de la implementación de MILENIO detalladamente, para ello se hará un acercamiento a algunos elementos básicos así como a la descripción de algunas clases con sus respectivos métodos para la comprensión total del funcionamiento del sistema.

17.1 ELEMENTOS DE UN AGENTE MÓVIL

En el centro de su estructura podemos encontrar una instancia del objeto *AgentRef*, el cual es un objeto compuesto de varios componentes adicionales además del propio agente. El objeto *AgentRef* se encuentra dentro del marco de trabajo de MILENIO y no esta disponible a los programadores. La estructura de *AgentRef* se observa en la figura 17.1.1.

El *AgentRef* puede considerarse como la implementación completa de un agente ya que se encarga de realizar funciones claves de un agente, incluyendo envío, recuperación y duplicado. Algunas invocaciones de métodos en el objeto agente son reenviadas al *AgentRef*. (Véase figura 17.1.1).

El objeto *AgentRef* es un objeto de referencia que es almacenado en la *AgentTable* (Tabla de referencias de Agentes del sistema). Esta tabla es usada para mantener una relación de todos los agentes que se encuentran en el sistema y para hacer búsquedas de ellos.

El campo de estado en la tabla se refiere a la situación actual del agente, será un 1 si se encuentra activo y un 0 si se encuentra inactivo (serializado) en un estado persistente.

El identificador de un agente debe ser único en cualquier sistema que de encuentre. Debido a esto no se pueden considerar nombres que proponga el usuario, ya que puede haber dos usuarios que propongan el mismo nombre en un sistema local o bien en sistemas diferentes en donde existiría el conflicto a la hora de la migración.

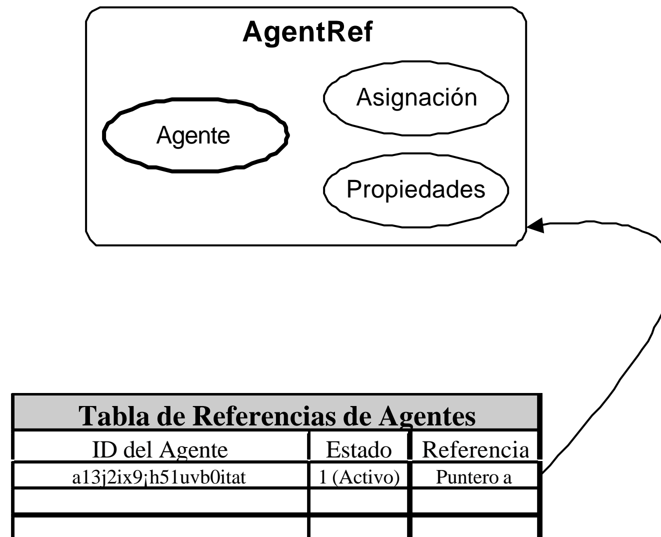


Tabla de Referencias de Agentes		
ID del Agente	Estado	Referencia
a13j2ix9¡h51uvb0itat	1 (Activo)	Puntero a

Figura 17.1.1 – Estructura del Objeto Agente

La dificultad en conseguir la unicidad del identificador se ve agrandada cuando consideramos que no podemos tomar el nombre de la clase de la que se origina el agente porque puede haber dos agentes en el mismo sistema que hayan sido creados a partir de la misma clase, inclusive puede haber dos servidores de agentes ejecutándose en el mismo ordenador.

Para conseguir la unicidad del identificador debe ser compuesto por varios elementos cuya suma total lo convierta en único. Es decir, elementos que considerando las situaciones más remotas nunca puedan coincidir en su conjunto, por ejemplo:

FechaHoraDeCreaciónDelAgente + Servidor + DirecciónIP

En donde *FechaHoraDeCreaciónDelAgente* será el parámetro incluido en las propiedades del agente (*CreationDate)* que incluye no sólo la fecha sino la hora detallando hasta milisegundos. *Servidor* es el nombre del servidor de agentes en donde se creó el agente y *DirecciónIP* es la dirección IP del ordenador donde se creó el agente.

Los últimos dos parámetros son necesarios ya que dentro de un mismo ordenador pueden crearse dos sistemas de agentes diferentes y porque podría darse el caso que dos agentes se crearan a la misma hora en dos ordenadores diferentes con los que no sería suficiente la fecha y hora de creación.

Sin embargo el problema no termina aquí, este identificador presenta dos inconvenientes: la longitud puede ser variable ya que el nombre del servidor puede ser

personalizado por el usuario, y a la hora de migrar el identificador del agente proporciona información que puede ser mal empleada por grupos terceros.

Para solucionar los inconvenientes se utiliza una función de conversión[90] de *N* a *M* para convertir una cadena variable de longitud *N* a una cadena de longitud fija *M*. Esta función tiene además las siguientes propiedades:

- Dadas dos cadenas diferentes de entradas jamas podrá generar dos cadenas de salida iguales con lo que se mantiene la unicidad de nuestros identificadores.
- No proporciona ninguna información sobre los datos personales del agente.
- El identificador es único y puede ser usado en cualquier sistema.

En el caso de MILENIO, para generar los identificadores de los agentes la función recibe una cadena de longitud variable y devuelve una cadena (*message digest*) de 20 caracteres de longitud[91].

Es importante señalar que hay varios sistemas de agentes que no logran la unicidad del identificador de sus agentes lo que generan muchos problemas al intentar hacer aplicaciones robustas, por ejemplo concordia[WALS98] hace una combinación de usuario@grupo que le permite identificar perfectamente al agente localmente pero que crea conflictos cuando migra ya que pueden haber coincidencias. Los *Aglets* utilizan un mecanismo parecido al de MILENIO y si logran la unicidad de los identificadores.

Volvamos a los componentes de *AgentRef*, el objeto de propiedades contiene los elementos de identificación que le son asignados al agente a la hora de su creación. En principio nos encontramos con un objeto del tipo *NameMobileAgent* que define las propiedades de identificación del agente y con un objeto *Itinerary* que define el recorrido que va a tener el agente[92].

```
NameMobileAgent  Name           // nombre del agente
Itinerary  Travel               // itinerario
```

En donde la clase *NameMobileAgent* contiene los siguientes atributos:

```
String AgentClass               // nombre de la clase a partir de la cual se creó
String AgentAuthority           // propietario del agente
NameAgentSystem  AgentSystem    // datos del sistema de agentes donde fue creado
Date CreationDate               // fecha y hora de creación del agente.
CodeSource codeSource           // sitio de origen del agente.
```

A su vez la clase *NameAgentSystem* define los atributos de identificación de un sistema, por lo que se incluyen los atributos:

```
String Authority                // propietario del sistema de agente
String AgentSystemId            // identificador único del sistema de agentes
String IP                       // dirección IP del sistema de agentes
```

[90] Un message digest con algoritmo SHA.

[91] Milenio no utiliza algunos otros algoritmos de conversión existentes en el mercado para generar los identificadores por que no se encuentran disponibles directamente en el lenguaje Java.

[92] El itinerario puede o no ser usado a la hora de su creación, es decir, puede haber agentes que no definan ningún tipo de itinerario.

En el momento de su creación el campo *Propiedades.Name.AgentAuthority* definirá la autoridad o dueño del agente. Basándose en esta campo se obtendrá de la base de datos *DBPreferences* un objeto llamado asignación de tipo *ResourcesAvailable* que indica las restricciones que tendrá el agente una vez que migre a un servidor remoto. Este objeto será igualmente asociado al *AgentRef* que evidentemente también será parte del agente.

Los campos de dicho objeto son los siguientes:

```
Int Disco              // Cantidad en kilobytes que puede usar
Int Hijos              // Número de hijos que puede crear el agente
Int Memoria            // Cantidad en kilobytes que puede usar
Boolean Mensajes       // Capacidad o no de enviar mensajes
Boolean Duplicar       // Capacidad o no de duplicarse
Boolean Share          // Capacidad o no de compartir ficheros
```

Cuando se utilice el método *clone* para duplicar agentes se utilizará el *AgentRef* para realizar la copia.

17.2 CREACIÓN E INICIACIÓN DE AGENTES

Como se puede apreciar en el diagrama de clases de la figura 16.2.4.1.1, existen varias clases involucradas en la creación de un agente, los pasos que se siguen para su creación e inicialización son los siguientes:

1. Se crea una instancia de la clase *AgentHandler* que será el encargado de crear, inicializar y ejecutar al agente.
2. El *AgentHandler* localiza la clase del agente, carga sus datos y define una nueva clase[93] (si no se encuentra ya definida).
3. Crea una nueva instancia del nuevo objeto de esa clase (ejecuta su constructor).
4. Crea un objeto de referencia *AgentRef* y establecer la conexión con su agente, sus propiedades y su asignación.
5. Comienza la ejecución del agente.

Obsérvese que la conexión entre el objeto de referencia y el agente no es creada hasta después que la ejecución del constructor del agente esta completa. La implicación es que el agente no puede acceder a ningún método de las clases del agente desde su constructor.

Es necesario evitar el uso del constructor del agente para inicializarlo, en su caso se sugiere usar el método *oncreation()*. Esto se debe a que el constructor sin argumentos de un agente que implementa *java.io.externalizable* se llama cada vez que el agente es deserializado, si el agente se inicializa en su constructor dicha inicialización de valores se volverá a llamar cada vez que se serialize/deserialize con lo que se perdería su estado de datos.

[93] En Java, definir una nueva clase significa cambiar un vector de bytes en una instancia de la clase *Class*.

17.3 SERIALIZACIÓN DE AGENTES

Cuando se envía un agente, se desactiva o se duplica, se realiza el proceso de serialización del agente mediante los métodos de la clase *serializer*. El objetivo de este proceso es conservar el estado actual de las variables del agente (estado de datos) ante una operación de envío o desactivación temporal.

Los métodos de la clase *serializer* que permiten serializar el estado de datos del agente son:

```
Void serialize(Object Agente, String Fichero)
Void serialize(Object Agente)
```

Al primer método se le pasa como parámetro además del agente el nombre del fichero que se va a generar para guardar el estado serializado. Al segundo solo se le pasa el nombre del agente ya que ya que la salida la dirige a un fichero del mismo nombre (con extensión *.os)*.
Y los métodos que permiten realizar la tarea inversa son:

```
Object de_serialize(String Fichero)
Object de_serialize(InputStream Entrada)
```

Ambos métodos devuelven el objeto deserializado el primero lo deserializa a partir del fichero que se le pasa como parámetro y el segundo a partir de la *InputStream* indicada.

El proceso de serialización comienza con el objeto del agente que contiene su estado de datos y recorre todos los objetos que estén al alcance, excepto aquellos que contengan referencias *transient.* Adicionalmente, las propiedades del agente, así como su asignación, son puestas en un vector de bytes. Este vector es usado para crear una copia, enviarlo a un almacenamiento persistente o para enviarlo a través de la capa de comunicación.

El entorno de ejecución de MILENIO usa el mecanismo de serialización estándar de Java para almacenar y recuperar el estado de datos de un agente a un fichero. Durante la serialización, todos los objetos *nontrasient* que son alcanzables desde el objeto del agente son considerados como su estado y son visitados para incluirlos en el fichero. Por lo tanto, todos los objetos que vayan a ser transferidos deben ser *nontransient* y deben de implementar la interfaz *java.io.Serializable* o la *java.io.Externalizable* para ser serializados apropiadamente. Si se encuentra un objeto que no sea serializable se generará una excepción.

Es importante observar dos cosas, los objetos gestionados en el proceso de serialización/ deserialización son almacenados por valor. Esto es, un objeto que es compartido por varios agentes deja de ser compartido una vez serializado, por lo que deja de ser compartido al ser enviado o desactivado. Por otro lado, los valores de las variables de clase (variables *static*) que no son parte del objeto nunca son serializados y por consiguiente tampoco son transferidos. De esta forma, las variables de clase, son locales a esa clase y el agente puede obtener un valor distinto cuando éste llegue a un nuevo destino y acceda a la variable de clase proporcionada.

17.4 MIGRACIÓN DEL AGENTE

Como describí anteriormente, un agente inicialmente esta compuesto por su clase principal (por ejemplo MyAgent.class) y un por un conjunto de clases requeridas por él mismo. De igual forma contiene un objeto de propiedades (que incluyen su autoridad, sistema de agentes, fecha de creación, etc.) y su asignación (la cantidad de recursos que puede consumir en sitios remotos y que le son concedidos por el administrador).

Cuando un agente va a migrar estos elementos son comprimidos en un fichero *Jar* de Java (MyAgent.jar) mediante los métodos de la clase *JarCompressor* del MILENIO. También es incluido en el fichero el estado serializado del agente (MyAgent.os) y su *CodeBase* que es usado por el cargador de agentes remoto en caso que quiera cargar una clase que no se encuentre dentro del fichero *Jar*. Una vez incluidos todos los archivos se firman todos los elementos del fichero con la firma de la autoridad del agente. Los ficheros de las firmas así como el archivo *manifest* usado para la gestión interna de las firmas son también incluidos en el fichero *Jar* antes de que éste sea enviado (figura 17.4.1).

El agente es convertido en un *Jar* para facilitar el futuro proceso de carga de clases en el servidor remoto y para que al ser comprimido su transmisión sea más rápida. Los métodos de la clase *JarCompressor* que permiten hacer esta tarea son:

```
void add(string Fichero)
void extract(string Fichero)
```

Ambos reciben el nombre del fichero que va a ser introducido al *Jar* o bien extraído del mismo.

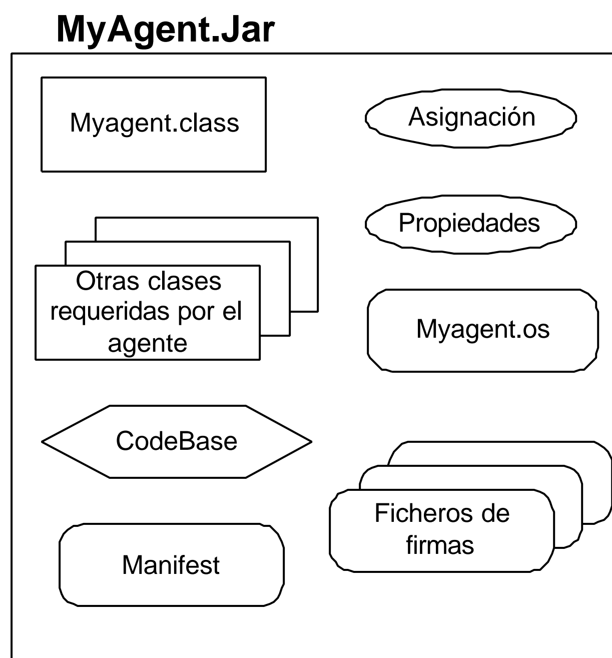


Figura 17.4.1 – Componentes del Agente al Migrar

Cuando un agente está en un servidor remoto y ha cambiado su estado de datos. El estado anterior es reemplazado por el nuevo estado de datos que se ha generado en dicho servidor, es decir, el fichero *MyAgent.os* es sustituido. Como el nuevo estado de datos ya no puede ser firmada por la autoridad original (porque ya no se encuentra en su servidor de origen) el nuevo estado de datos es firmado por la autoridad del sistema remoto. El propósito de esta firma es de adjudicarle responsabilidad al servidor remoto en caso de que alteraciones maliciosas introducidas en los datos del agente produzcan un mal comportamiento del agente en el siguiente servidor. O bien garantizar que el no ha alterado la integridad de los datos del agente. En otras palabras la firma del servidor significa: "*Yo no he alterado la integridad de los datos de este agente*".

Esta manera de enviar comprimido y firmado al agente es una característica particular de este sistema ya que ninguno de los sistemas actuales incorporan esta característica, el sistema de los *Aglets* fue el primero en plantear la posibilidad de comprimir el agente en un *Jar* para acelerar su transmisión y por todas las ventajas que esto ofrece, no obstante no lo han implementado hasta ahora. Las últimas versiones disponibles de Concordia[WALS98] *D'Agents* y *Ara* no incluyen ningún mecanismo de compresión en las versiones actuales.

17.5 TRANSFERENCIA DE CLASES

En los sistemas de agentes móviles, las clases que un agente requiere para ser instanciado necesitan estar disponibles en el servidor remoto en donde se supone que se va a ejecutar. Consecuentemente, los sistemas de agentes deben cargar el bytecode requerido en demanda desde el servidor remoto o bien transferir el bytecode con el mismo agente.

A la hora de decidir cual de las dos técnicas implementar es importante considerar las ventajas que cada una puede tener. La primera en teoría agiliza el envío del agente, pero cada vez que requiera una clase para crear la instancia de un objeto del agente, el sistema perderá mucho tiempo en abrir una nueva conexión, solicitarla y traerla. El segundo esquema evita las perdidas innecesarias de tiempo al requerir datos de la red, pero corre el riesgo de sobrecargar el agente con clases que quizá en un futuro no llegaría a utilizar[94].

Es muy importante para el sistema la transferencia óptima de las clases, es fundamental saber decidir que clases deben ser transferidas y cuales no, así como saber cuales son parte del sistema local y cuales han sido definidas por el usuario. Las clases pueden agruparse en tres categorías de acuerdo a su origen, cada grupo recibe un tratamiento específico. Los grupos son los siguientes:

- Clases comprimidas en un fichero *Jar* que fueron definidas por el usuario y que son cargadas desde el fichero.
- Clases cargadas del *codebase* de procedencia del agente.
- Clases del sistema y que son cargadas automáticamente del *Classpath.*

[94] Como aquellas que han sido definidas dentro de los procedimientos

A la hora de decidir que clases son transferidas es necesario identificar a que grupo pertenece cada clase. El sistema de MILENIO al igual que los *Aglets* combina los dos esquemas mencionados anteriormente para transferir las clases requeridas de la forma más flexible. La mayoría de los sistemas desarrollados en Java[95] envía las clases en demanda una vez que son solicitados desde el servidor remoto lo que facilita el envío.

El sistema de los *Aglets*[LANG98b] al igual que *Mole*[HOHL98] poseen un mecanismo más sofisticado para la gestión de las clases, incluyen una cache que se encarga de buscar en las clases llevadas al sistema por otros agentes antes de irlas a buscar al servidor remoto.

El único problema que se presenta con este mecanismo y que es muy difícil de solucionar es el problema de las versiones de clases, es decir, un agente trae una clase al sistema llamada *Agenda* desde un sistema X, posteriormente otro usuario de dicho sistema crea otro agente que usa la misma clase *Agenda* pero modificada. Cuando el agente llegue al sistema remoto cargada la clase *Agenda* antigua y se generará un conflicto. Debido a la dificultad para resolver este proble decidí no implementarlo hasta que Java ofrezca una forma de etiquetar las clases con una versión específica.

17.5.1 Clases transferidas con el agente

La forma más eficiente de transferir clases consiste en enviar con el agente todas las clases requeridas para su creación y después enviar el resto de las clases si son solicitadas en tiempo de ejecución y no existen en el servidor remoto.

MILENIO envía comprimidas todas las clases requeridas para la creación del agente en el fichero *Jar* (figura 17.4.1). Sólo comprime y envía las clases públicas que son definidas por el usuario y que no forman parte del sistema.

Si una clase es requerida para la creación del agente pero forma parte de Java, la clase no será transferida ya que se cargará automáticamente desde el *Classpath* del sistema remoto. En otras palabras, sólo se envían las clases que hayan sido definidas como públicas por el usuario y que no se encuentren dentro del fichero *rt.jar* en el cual se encuentran comprimidas todas las clases del Java.

En el caso de que todos los servidores cuenten con una librería o extensión adicional, será posible que el núcleo del sistema a la hora de decidir que clases envía, verifique la existencia de las clases no sólo en el fichero *rt.jar* sino también en el fichero comprimido en donde se encuentran las clases de la extensión.

[95] Los sistemas como D'Agents cuya plataforma es *TCL* no se enfrenta al problema de envío de clases ya que es problema es particular a los sistemas basados en Java.

17.5.2 Clases requeridas desde el lugar de procedencia del agente

Si el agente tiene métodos que hagan referencia a clases definidas por el usuario pero que no son públicas o necesarias para su creación no serán transferidas inicialmente.

Si durante su recorrido el agente requiere ejecutar ese método y por consiguiente le es indispensable la existencia de dicha clase en el servidor remoto en el que se encuentre, dicho sistema se la pedirá al servidor de origen, es decir, el servidor remoto las requerirá en demanda al servidor de origen para que estas puedan ser cargadas en el servidor remoto.

Para lograr esta trasferencia el cargador de agentes (*AgentLoader*) creará una instancia de la clase *GetClass* que se encargara de contactar al sistema de origen estableciendo una conexión SSL, el sistema de origen creará un objeto *SendClass* que se encargue de enviar las clases que le son demandadas a través de la conexión SSL.

Las clases que sean enviadas desde el sistema de origen y que sean cargadas posteriormente en el sistema remoto gozarán de los mismos privilegios que le habían sido concedidos a la autoridad propietaria del agente al que pertenecen.

17.5.3 Clases del sistema cargadas desde el classpath

Como mencione anteriormente, sí se requiere una clase para crear un agente o bien para invocar alguno de sus métodos, y esta clase pertenece a Java o bien a alguna de sus extensiones[96] definidas en el sistema, ésta se cargara en forma automática desde el *classpath*, desde */jre* (directorio donde normalmente se guardan las extensiones) o desde cualquier otra ruta definida en el sistema.

Cuando un agente requiere una clase, el sistema en forma automática se encarga de verificar si a clase existe en el fichero *jar* que representa al agente y en el cual se encuentran comprimidas sus clases, si no la encuentra verifica en el *classpath* y en todas las direcciones que se le hayan especificado al sistema, finalmente, si no la encuentra en ningún sitio dentro del sistema local, la solicita al servidor de origen.

17.6 PROTOCOLO DE TRANSFERENCIA DE AGENTES

Cuando un método ejecuta el método *dispacth()*, el gestor de servicios *AgentServices* se encarga de procesar su transferencia. El fichero *Jar* que representa al agente es transferido mediante *sockets*, por debajo se encuentra el protocolo SSL para garantizar una completa seguridad en la transferencia. Mediante este protocolo los servidores se autentifican mutuamente antes de iniciar una conexión. La figura 17.6.1 muestra las clases que intervienen en el envío y recepción de un agente:

[96] Por ejemplo librerías de criptografía incorporadas adicionalmente a Java2 SDK, necesarias para todas las tareas de encriptación y transferencia que realizan los agentes.

Para enviar al agente *AgentServices* invoca métodos de *Serializer* para serializar el estado de datos del agente, de *JarCompressor* para comprimir todos sus objetos (*allowance* y *properties*) y clases dentro de un fichero Jar y de *Signer* para firmar el agente con la firma de la autoridad a la que pertenece. Enseguida crea una instancia de la clase *Sender* a la que le pasa el identificador, destino y nombre de la clase principal del agente, posteriormente *sender* se encarga de contactar con el servidor remoto, de iniciar una conexión SSL mediante *SSLSender* y de hacer el envío enviando todos los datos de uno a uno.

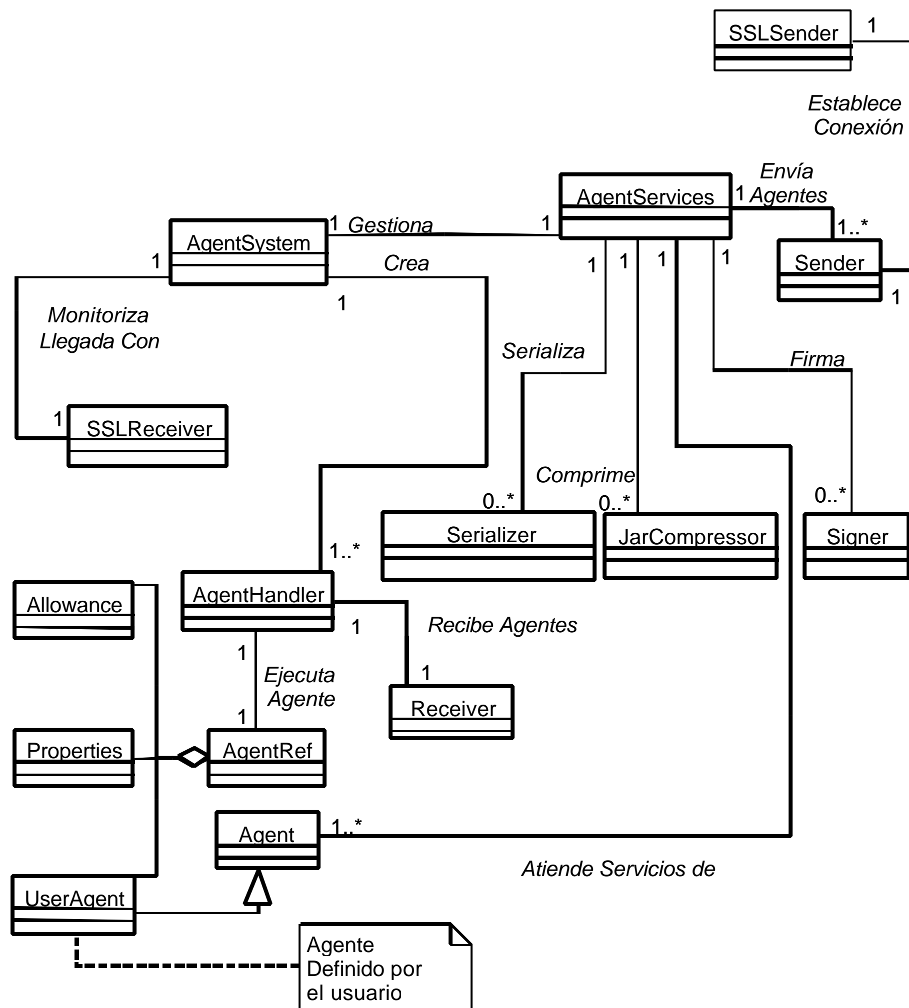


Figura 17.6.1 – Clases que intervienen en el Envío/Recepción de un Agente

SSLSender crea un *socket* SSL que se encarga de hacer el *handsake* (saludo) y autenticación entre los servidores. Después transmite el fichero Jar que contiene al agente comprimido y firmado, para ello envía primero su nombre y tamaño.

Por su parte el servidor escucha permanentemente mediante *SSLReceiver*, cuando recibe una señal de transferencia invoca el método *tratarconexion()* quien se encarga de crear un *AgentHandler* que será el encargado de reanudar al agente.

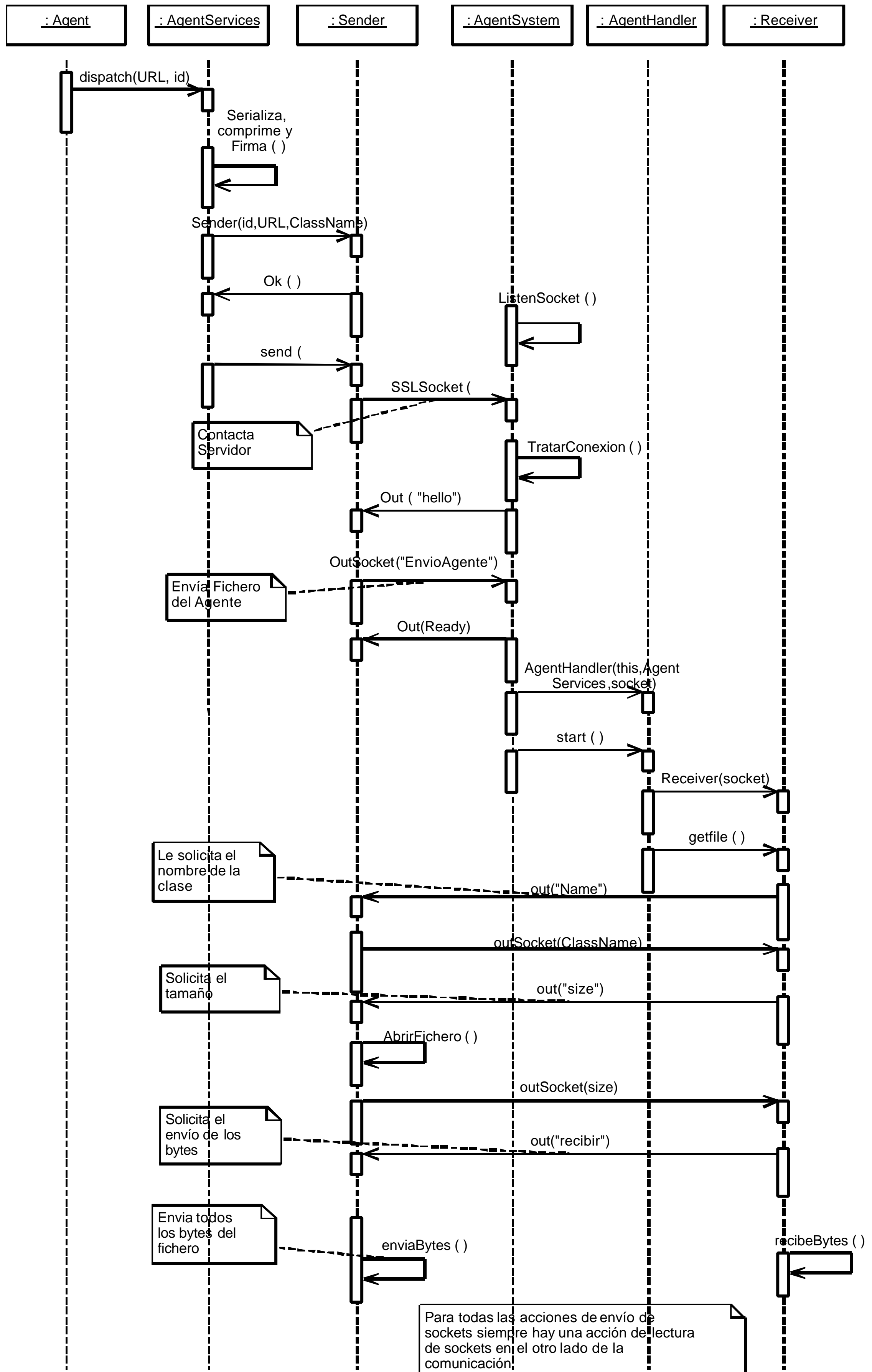


Figura 17.6.2 – Protocolo de Transferencia de Agentes

El *AgentHandler* su vez creara un receptor *Receiver* para que reciba el fichero *Jar* que contiene al agente, posteriormente invocará al *AgentLoader* para que lo cargue y realice diversas verificaciones de seguridad, finalemente creará un hilo y ejecutará al agente. Los detalles de protocolo de transferencia se observan en la figura 17.6.2 (no se incluyen las clases en el diagrama de *serializer*, *JarCompressor* y *signer* por cuestiones de espacio).

Una versión reducida de este protocolo es empleada para transferir las clases que no se envían con el agente cuando son requeridas por un sistema remoto que intenta reanudar el agente. Esta transmisión de datos como lo mencionamos anteriormente lo llevan a cabo *GetClass* en el servidor remoto y *SendClass* en el servidor de origen.

Capítulo XVIII
MILENIO: DESCRIPCIÓN DE LA IMPLEMENTACIÓN DE LA ARQUITECTURA DE SEGURIDAD SAHARA

Dentro de este capítulo describiré todos los detalles de implementación de la arquitectura de seguridad SAHARA dentro del sistema de Milenio. Adicionalmente describiré los diversos mecanismos de seguridad con que cuenta el sistema para protegerse así mismo contra intrusos, así como para proteger los repositorios de claves y ficheros de políticas de seguridad.

18.1 SEGURIDAD EN EL ACCESO AL SISTEMA Y EN LA ASIGNACIÓN DE SUS RECURSOS

Dentro del proceso de instalación de Milenio se definirá una cuenta de administrador que será el único habilitado para crear las cuentas del resto de los usuarios. Una vez terminada la instalación del sistema al arrancarlo aparecerá una ventana de acceso en donde todos los usuarios tendrán que teclear el nombre de su cuenta y su clave de acceso los cuales serán validados contra el repositorio de claves del sistema llamado *Keystore*.

El administrador gozará de privilegios especiales que no tendrán el resto de los usuarios/autoridades del sistema. Por ejemplo, podrá definir diversas opciones de la configuración del sistema entre ellas la forma en que los agentes transferirán las clases requeridas para ser instanciados en el servidor remoto.

Sin embargo, una de sus funciones principales será la de crear las cuentas del resto de los usuarios del sistema así como de asignar la cantidad de recursos que podrán consumir las autoridades dentro del sistema local y en los sistemas remotos.

Al dar de alta un usuario local se especificará (además de los datos de la cuenta) su asignación, es decir, la cantidad de recursos[97] que cada agente de dicha autoridad podrá consumir en un sistema remoto (véase figura 18.1.1). La cuenta y su clave de acceso así como las claves pública y privada que creará el sistema para él serán guardadas en el *keystore* mientras que su asignación será guardada en el fichero *DBPreferences*.

Figura 18.1.1 – Definición de asignaciones para los usuarios locales del sistema

Por último la posibilidad de clonarse y la posibilidad de enviar mensajes[98] son guardadas también en su objeto asignación, la posibilidad de clonarse será revisada antes de que el sistema le permita a un agente hacer una copia.

A su vez cuando se agrega una autoridad remota, inicialmente se importa el certificado asociado con su clave pública y que es necesario para comprobar los agentes firmados por él. Los elementos de la cuenta y su certificado se dan de alta en el *Keystore*, dicha entrada sólo servirá para validar los agentes provenientes de servidores remotos. Como dicha autoridad no entrara en el sistema localmente si no sólo a través de los agentes que envíe, no es necesario asignarle un password.

Por otro lado, cada vez que se incorpora una autoridad remota al *Keystore* es necesario definir la cantidad global de recursos que los agentes pertenecientes a esa autoridad podrán consumir (Figura 18.1.2). Esta cantidad representa el límite para la suma total de las asignaciones de los agentes pertenecientes a dicha autoridad y que se ejecuten localmente en el sistema. Las cantidades serán guardadas en el fichero *DBResources*.

Por último el certificado lo podrá obtener a través de un fichero .cert o bien por medio de un Jar desde cualquier parte del sistema de ficheros e inclusive a través de una unidad o conexión de red.

[97] Como se describió en capítulos anteriores dentro de estos recursos se especifica el número de hijos que un agente puede tener, el número de conexiones de red que puede establecer y las cantidades de disco duro y RAM que pueden consumir aunque estas dos últimas no se implementan en esta versión porque la versión actual de Java no lo permite.
[98] Esta opción será considerada en la siguiente versión de MILENIO.

Figura 18.1.2 – Definición de asignaciones para las autoridades/usuarios remotos

De la misma manera que se dan de alta las asignaciones se dan de alta los permisos sobre el sistema de ficheros y conexiones de red, para ello, para cada autoridad local y remota el administrador definirá una entrada *grant* en el fichero de políticas de seguridad en donde se establecen todos sus privilegios(rutas, ficheros, tipos de acceso).

Ningún usuario que no sea el administrador podrá modificar las cantidades asignadas para cada autoridad ya sea local o remota, ni en el fichero de políticas de seguridad ni en el fichero que define las asignaciones.

Debo señalar que actualmente ningún sistema de agentes móviles ofrece seguridad en el acceso al sistema y en la asignación de recursos, es decir, en otros sistemas cualquier usuario puede modificar las políticas de seguridad del sistema lo que afecta directamente a la asignación de recursos.

El verificar el acceso e incluir un administrador del sistema permite una mejor distribución de recursos y sobre todo ofrece una garantía adicional de que nadie incluirá usuarios no deseados ni modificará las políticas de seguridad vigentes acordes con los intereses de la empresa que representa. De igual forma permite asignar la responsabilidad del buen funcionamiento del sistema sobre el administrador, ya que si cualquier usuario pudiera modificar la configuración del sistema, en caso de fallo no se podría adjudicar la responsabilidad del mismo a ninguna persona.

18.2 SEGURIDAD DEL REPOSITORIO DE CLAVES Y DE LOS FICHEROS DE SEGURIDAD

Cada entrada en el repositorio de llaves llamado *keystore* se protegerá con una clave de acceso la cual será requerida cada vez que se acceda a la firma digital de la autoridad que se encuentra en ese registro. A su vez el *keystore* en su conjunto será protegido con

un clave de acceso adicional única que será proporcionada por el administrador durante el proceso de la instalación, de está manera se logra una doble protección de las claves evitando alguna posibilidad de uso indebido.

El núcleo del sistema se encargará transparentemente de proporcionar al sistema de seguridad de Java las dos claves de acceso requeridas para acceder al registro de una autoridad cuando sea necesario.

Para acceder a la llave general del *keystore* el núcleo al arrancar el sistema desencriptará la llave única proporcionada por el administrador desde el fichero oculto *password.key*. Para lograr transparencia y sencillez en la gestión y validación de la segunda clave de acceso, el núcleo del sistema hace corresponder la clave de acceso individual para cada autoridad en el *keystore* con la clave de acceso de esa autoridad al sistema de agentes.

De hecho, para que un usuario pueda acceder al sistema debe proporcionar esta clave de acceso la cual le será validada directamente en el *keystore*. Si el usuario/autoridad no proporciona la clave correcta le será denegado el acceso, en caso contrario el usuario podrá entrar al sistema y el núcleo contará con la segunda llave requerida por el sistema de seguridad de Java cuando dicho usuario/autoridad firme algún agente y el núcleo requiera su firma digital.

Actualmente los pocos sistemas de agentes que han incorporado las firmas digitales como mecanismo de autenticación de autoridades de los agentes (por ejemplo Concordia), no cuentan con un repositorio de llaves y certificados de donde el sistema de seguridad puede seleccionar el certificado de llave pública para comprobar la firma de un agente y con ello su autoridad.

Estos sistemas normalmente envían la llave pública adjunta con el agente para su verificación, la idea de crear un repositorio de llaves protegido es incluida en el sistema de seguridad de Java2 razón por la que todos los sistemas que han sido implementados hasta ahora usando el JDK 1.1.6 carecen de él. MILENIO es el primer sistema de agentes implementado con el JDK 1.2.1 el cual es llamado por *Sun Microsystems* Java2.

Para proteger los ficheros que definen las características de seguridad del sistema, (tanto los de políticas de seguridad como los de asignaciones) durante el proceso de instalación se guardarán dentro del directorio *$Home/Sahara/config*, al terminar dicho proceso al directorio se le restringirán sus permisos[99] de acceso de tal forma que sólo el administrador del sistema pueda acceder a ellos desde dentro y fuera del sistema de agentes.

Es importante señalar que dentro de estos ficheros se definen los privilegios de acceso tanto para autoridades/usuarios locales como remotos, por lo que la única forma de garantizar su integridad es permitiendo que sólo el administrador añada, modifique y restrinja dichos privilegios conforme a las necesidades del sistema de agentes.

[99] Los permisos de sistema operativo de escritura y modificación.

18.3 EL PROCESO DE FIRMADO DE UN AGENTE

Para que un sistema de agentes pueda aceptar agentes remotos es necesario que éste sea capaz de autentificarlos, para ello es necesario que el agente utilice un mecanismo por medio del cual pueda asegurar que él pertenece a la autoridad que dice y que posteriormente el sistema remoto pueda corroborar esa afirmación.

La autenticación es fundamental ya que basándose en la autoridad a la que pertenece el agente le serán concedidos los privilegios. En el capítulo XII se describió el proceso de autenticación de SAHARA que utiliza firmas digitales con distribución de llave pública.

Las firmas digitales además de permitir la autenticación proveen una protección adicional contra la integridad del agente y sus datos, ya que una vez firmado, si el agente o sus datos son modificados la firma ya no será válida y el sistema remoto podrá rechazarlo antes de que ingrese al sistema. De esta manera, las firmas digitales sirven para autentificar la autoridad de los agentes y para garantizar que ningún servidor malicioso o grupos terceros modifican al agente o a sus datos.

Actualmente los pocos sistemas que utilizan firmas digitales para la autenticación de los agentes no cuentan con un mecanismo bien diseñado para su verificación. Concordia envía el agente firmado con su certificado de clave pública adjunto para su verificación, lo cual ralentiza el proceso. *D'Agents* utiliza el mismo mecanismo e inclusive invoca un programa externo al sistema de agentes que se encarga de encriptarlos usando PGP. En MILENIO el proceso de firmado y validación utilizan el repositorio de llaves *Keystore* lo que facilita y acelera el proceso.

Para llevar a cabo el proceso de firmado del agente es necesario realizar varias tareas que permiten al sistema tener todos los elementos a la hora de firmar al agente. La figura 18.3.1 muestra el proceso y los elementos que intervienen en el proceso de firmado de un agente.

18.3.1 Alta de autoridades/usuarios y generación de firmas

Para que el sistema pueda firmar agentes es necesario que la autoridad local que esta enviando el agente tenga una clave privada en el *Keystore* que será usada para firmar el agente. Para ello cada vez que se dé de alta un usuario el *KeystoreManager* le solicitará al *KeyGenerator* un par de claves que serán guardadas en el *Keystore* una vez que la clave pública haya sido certificada. Cada entrada en el *Keystore* estará protegida por una clave de acceso.

La idea de asociar el nombre de la autoridad con el nombre de la cuenta de acceso al sistema, y de que su clave de acceso coincida con la que tiene asignada su registro en el *keystore* es una característica propia de MILENIO que permite una sola validación y un doble mecanismo de seguridad[100]. La mayor parte de los sistemas de agentes no validan el acceso al sistema.

[100] Esto se debe a que sólo se pide el nombre de la cuenta y la clave de acceso una vez y sin embargo se utiliza dos veces para validar su acceso al sistema y para acceder a los datos del *Keystore*.

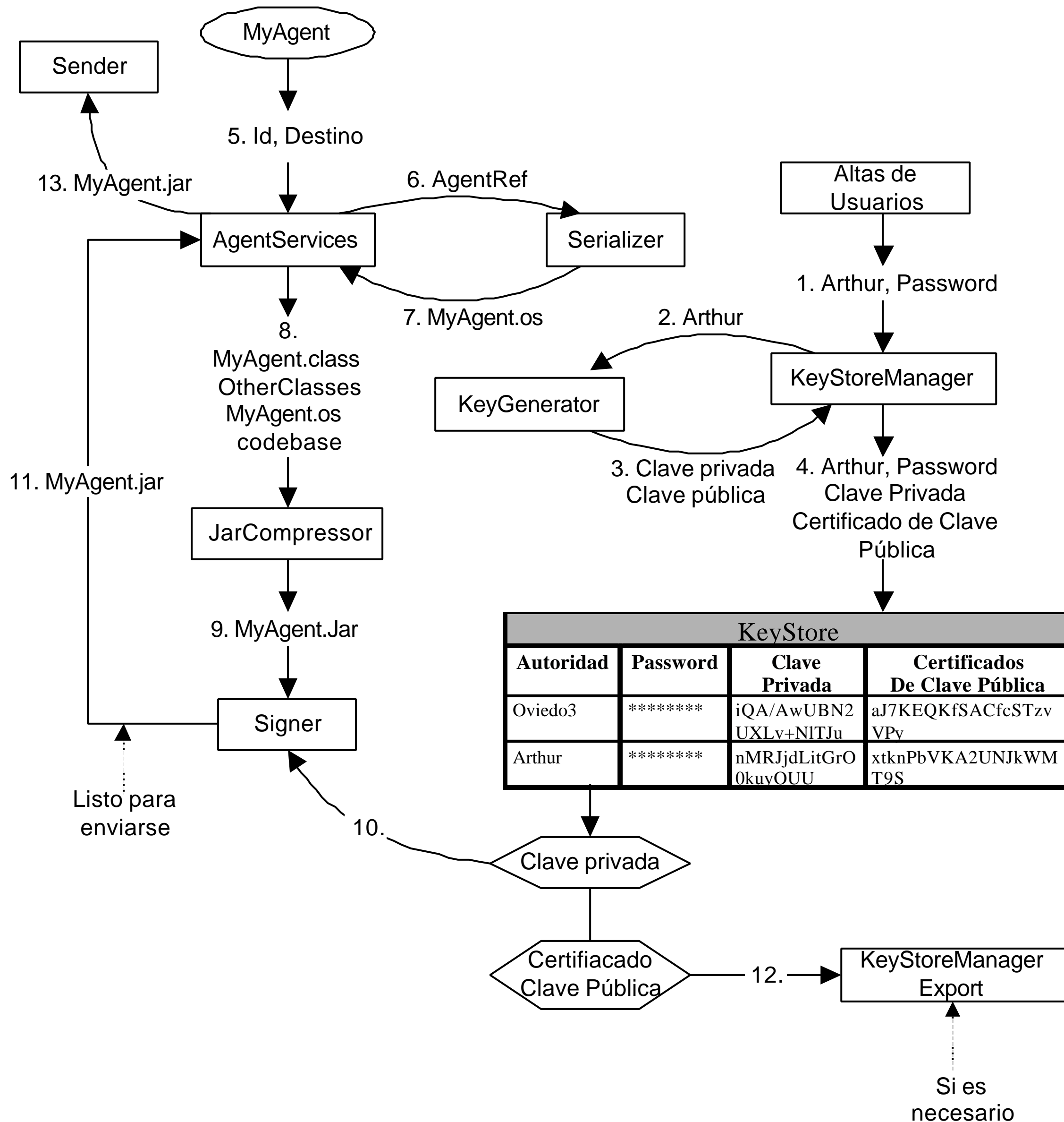


Figura 18.3.1 – Proceso de Firmado de un Agente Móvil

18.3.2 Preparación del agente para el envío

Cuando el agente decide migrar a otro servidor se lo comunica enviándole su referencia (*AgentRef)* a *AgentServices* quien es el encargado de preparar al agente para su envío. Para describir los pasos que se realizan se usará un ejemplo ilustrativo suponiendo que el agente que va a migrar se llama *MyAgent*.

Lo primero que se realiza es invocar a *Serializer* para que serialice el estado de datos del agente el cual será devuelto en el fichero *MyAgent.os*. Enseguida *AgentServices* le envía a *JarCompressor* la clase a partir de la cual se creo el agente así como otras clases que sean necesarias para su creación en el servidor remoto (ficheros .class), envía también el

estado serializado del agente (.os) y su *codebase*, todos estos ficheros son comprimidos en un fichero *MyAgent.jar* que posteriormente será firmado.

La compresión del agente tiene como finalidad acelerar el tiempo de envío ya que se acortará el tiempo de firmado, de transferencia, de validación y de carga del agente en el servidor remoto. La gran mayoría de los sistemas de agentes actuales no comprimen el agente y sus clases lo que ralentiza su envío. Concordia y el sistema de los *Aglets* lo incluirán en sus próximas versiones.

18.3.3 Proceso de firmado

Una vez que se tiene el fichero *MyAgent.jar* el sistema se encargará de firmar los datos para ello obtiene del *Keystore* la clave privada de la autoridad correspondiente, una vez generadas las firmas los ficheros de cada una de ellas así como un fichero llamado *manifest* que sirve para la gestión interna de las mismas son también incluidas en *MyAgent.jar*.

Para acceder a la clave privada de la autoridad en el *Keystore* el sistema se encarga de pasarle la clave de acceso correspondiente. Para la validación de la firma en un sistema remoto es necesario que el sistema haya exportado previamente el certificado correspondiente a la clave pública de dicha autoridad.

Una vez firmado el agente no es posible hacerle ninguna modificación ni a sus clases ni a su estado de datos y el fichero del agente comprimido y firmado es pasado a *AgentServices* para que posteriormente sea enviado.

Cuando un agente migra a un servidor remoto, normalmente su estado de datos cambia, con lo que ya no es válida la firma para el fichero correspondiente al estado de datos. Para resolver este problema, el sistema remoto sustituye en el fichero original del agente el nuevo estado de datos firmado por él mismo. Al imprimir su firma sobre el nuevo estado de datos el sistema remoto garantiza que él no ha alterado los datos que contenía el agente[101].

18.4 PROCESO DE ENVÍO Y SEGURIDAD EN LA TRANSMISIÓN

Es importante que después de firmar el agente el envío a su destino sea bajo un protocolo confiable que garantice su integridad y que evite los ataques de terceras partes o los espías en la red.

[101] Esta medida es utilizada como una forma de protección del agente contra servidores maliciosos, aunque no garantiza que se pueda realizar un daño, cuando menos se puede utilizar para adjudicar responsabilidades sobre el servidor malicioso.

Para ello, después de firmar el agente *AgentServices* se lo envía a *Sender* para que lo envíe al destino especificado. *Sender* a su vez creará una instancia de *SSLSender* que iniciará una conexión SSL con el servidor remoto.

Para establecer la conexión *SSLSender* realiza el protocolo de saludo (*handsake)* y la autenticación del servidor remoto. Una vez identificados mutuamente ambos sistemas definen un par de claves que serán utilizadas para encriptar y desencriptar la información.

El protocolo SSL definido en SAHARA es completamente estándar, en el cual para cada transferencia de información, los datos se encriptan en el servidor emisor y se desencriptan en el servidor receptor. Si algún bloque de información es bloqueado por un grupo tercero al no llegar completa a su destino es detectada por el servidor remoto, de igual forma el robo de información codificada no representa ningún problema ya que es imposible que tenga algún sentido sin la clave que la decodifica.

Como ya se describió en el capítulo XVII el protocolo de transferencia de agentes utilizado por MILENIO envía en varias fases todo el agente, primero envía su nombre, luego el tamaño, etc. A su vez el protocolo SSL para enviar la información abre un puerto a través del cual enviará la información codificada en forma de *sockets*. Dentro de una máquina servidora puede haber varios sistemas de agentes móviles activos a la vez, por lo que, para que envíen información a manera *sockets* al mismo tiempo es necesario que cada uno transmita por puerto distinto.

MILENIO implementa la especificación de SSL versión 3 (SSLv3) en Java puro y la integra al sistema, el único sistema que hace uso del mismo protocolo para ofrecer seguridad en la transmisión es Concordia y Ara lo incorporará en su próxima versión. El resto de los sistemas no ofrecen un mecanismo para proteger la transmisión.

18.5 PROCESO DE CARGA, AUTENTICACIÓN, ASIGNACIÓN DE PRIVILEGIOS Y DOMINIOS DE PROTECCIÓN.

Cuando un agente llega a un sistema remoto, éste lleva a cabo un largo proceso para cargarlo, autentificarlo y para asignarle sus privilegios asociándolo al dominio de protección de su autoridad o creándole uno nuevo en caso de que no exista.

Todas las tareas deben realizarse en secuencia ya que es necesario cargar para autentificar y esto a su vez para asignarle sus privilegios. Los dominios de protección se gestionan en conjunto a los del sistema de seguridad de Java. La gráfica 18.5.1 describe los pasos más importantes de este proceso ya que algunos fueron excluidos por cuestiones de espacio. Enseguida se explica en detalle cada uno de estos pasos.

18.5.1 Requerimientos previos a la carga del agente

Antes de que el agente pueda ser cargado y posteriormente validado es necesario que el gestor del *Keystore* del sistema remoto haya importado el certificado de clave pública

de la autoridad que se desea validar, adicionalmente deben incluirse sus permisos en los ficheros de políticas de seguridad del sistema así como dar de alta sus asignaciones.

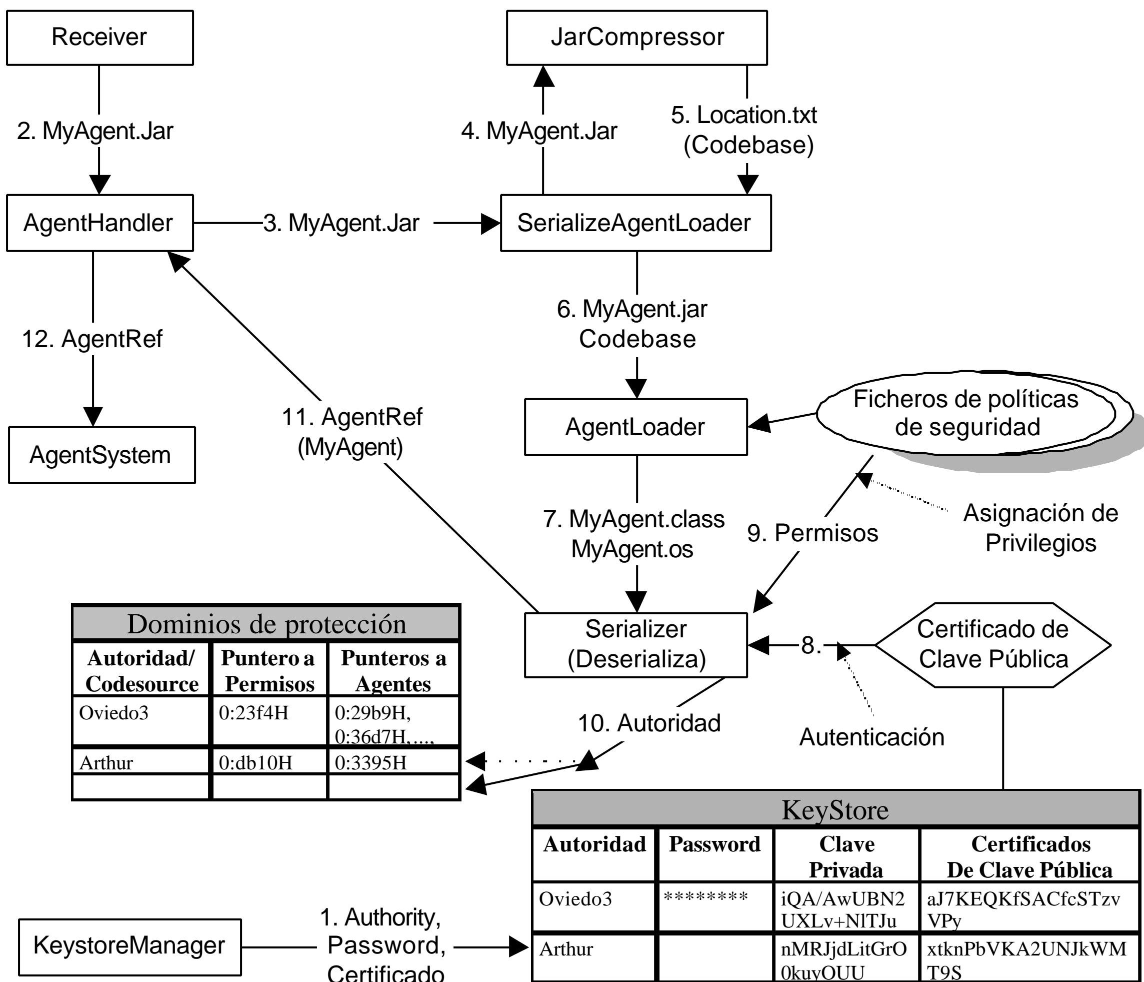


Figura 18.5.1 – Proceso de Carga, Autenticación y Asignación de privilegios de un Agente Móvil

Adicionalmente a esto el agente comprimido en un fichero *Jar* será recibido por *Receiver* desde el sistema remoto y será pasado a *AgentHandler* que se encargará de gestionar todas las tareas necesarias para reanudar su ejecución. Inicialmente enviará el fichero del agente a *SerializedAgentLoader* que se encargará de extraer su *codebase* y de prepararlo para su carga.

18.5.2 El proceso de carga de un agente móvil y el *AgentLoader*

Para cargar un agente se requiere crear un cargador de clases para cada uno llamado *AgentLoader* que será el encargado de cargar todas las clases que requiera el agente a la hora de su creación y durante su ejecución.

El *AgentLoader* actuará como un monitor cuya función será proveer al agente de las clases requeridas siempre que las requiera. Inicialmente el *AgentLoader* le pasa a

Serializer la clase principal del agente y su estado de datos para que los deserialize y restablezca su estado anterior en una instancia del agente.

Si al crear la instancia *Serializer* requiere de clases adicionales se las pedirá a *AgentLoader*. Cuando le sea solicitada una clase el cargador de clases *AgentLoader* irá a buscarla al fichero *Jar* del agente en donde se encuentran comprimidas sus clases más importantes, si no la encuentra irá a buscarla al *classpath* de Java y si tampoco la encuentra la solicitará al sistema remoto de donde proviene el agente Para ello creará una instancia de *Getclass* que se encargará de iniciar una conexión con el servidor remoto y traer la clase solicitada.

Las clases pueden ser demandadas a la hora de su creación o posteriormente cuando un agente invoque un método que contenga una clase que no se encuentre cargada en memoria.

El *AgentLoader* a la hora de ser creado recibe de *SerializedAgentLoader* el *codebase* que utilizará para ir a buscar las clases que se requieran cuando no se encuentren en el sistema local.

18.5.3 Autenticación del agente

Cuando *serializer* recibe el estado de datos y la clase del agente crea una instancia de esa clase sobre la que leerá el estado deserializado de datos. A la hora de cargar la clase para crear la instancia se efectúa la validación de la firma digital, para ello el *Keystore* proporciona el certificado de la clave pública de la autoridad a la que pertenece el agente, si alguien alteró o modificó alguna parte del agente la firma no será válida y el agente será rechazado del sistema.

Si la firma es válida se tiene la garantía que dicha autoridad envío y firmó ese agente y de que no ha sido alterado por nadie, de esta forma podemos confiar plenamente a la hora de otorgar permisos a los agentes de dicha autoridad.

18.5.4 Asignación de privilegios asociando a dominios de protección existentes

Una vez que se conoce la autoridad del agente, *Serializer* apoyado por el sistema de seguridad de Java verifica si ya existe un dominio de protección creado con sus respectivos permisos para esa autoridad, en caso de que así sea, bastará con asociar al agente al dominio de protección existente de esa autoridad. De esta forma el agente gozará de los permisos que le fueron otorgados a su autoridad.

Es muy importante señalar que cuando un agente móvil regresa a su sitio de origen normalmente existe el dominio de protección que había sido creado para su autoridad inicialmente y que corresponde a su *codebase*. Es decir, cuando un agente se crea localmente también se le asignan privilegios, sin embargo como a la hora de su creación no se firman, los privilegios no pueden ser asignados en función de su firma/autoridad.

Para solucionarlo, cada autoridad/usuario del sistema tendrá asociado un codebase, (un directorio donde se crean todos los agentes de esa autoridad). Para concederle permisos en las entradas *grant* de los ficheros de políticas de seguridad, es posible también la asignación de privilegios basándose en un *codebase*, por lo que se toma el *codebase* asociado con cada autoridad para otorgarle privilegios. Todos los agentes creados desde ese *codebase* gozarán de los mismos privilegios y serán incluidos en un mismo dominio de protección.

Cuando un agente local migra a otro servidor se firma digitalmente con la firma de la autoridad que le corresponde, y se le asignan privilegios en función de esa firma. Sin embargo cuando regresa a su sitio de origen, al ser autentificado e identificado como agente local, se asocia al dominio de protección que se había creado anteriormente basándose en el *codebase* de la autoridad.

La asignación de privilegios utilizada en MILENIO en función de una autoridad o de un *codebase* permite granularidad y sencillez en la asignación de privilegios, más aún, al ser un mecanismo integrado al sistema de seguridad de Java facilita el cumplimiento de privilegios de cada agente en el sistema ya que la verificación se realiza automáticamente desde el interior de la máquina virtual.

La mayor parte de los sistemas asignan privilegios usando ACLs (Listas de control de Acceso) como es el caso de *D'Agents* o de Concordia que lo hace asignando a cada autoridad identificada una serie de permisos que previamente habían sido definidos. *Ara* por su parte lo hace a través de un objeto llamado asignación que se le asigna al agente en su entrada y que no es más que un objeto que define sus permisos y que resulta ser similar a la ACL´s. Ambos mecanismos presenta la desventaja que para cada acceso que desee hacer un agente hay que realizar una gran cantidad de verificaciones afectando su desempeño. En el caso de MILENIO solo es necesario identificar su dominio de protección y verificar la correspondencia del permiso solicitado.

Los *Aglets* por su parte sólo definen dos grandes grupos, *trusted* que son los agentes locales y pueden acceder libremente a todos los recursos del sistema y *untrusted* que son los remotos que solo podrán acceder a los recursos que defina el administrador. Evidentemente este método presenta una gran carencia de granularidad a la hora de asignar privilegios, ya que las distintas autoridades de los agentes remotos son tratados de la misma forma. En MILENIO se pueden otorgar privilegios particulares a cada autoridad local y remota.

18.5.5 Asignación de privilegios creando nuevos Dominios de Protección

Si no existe un dominio de protección creado para la autoridad del agente entonces se inicia una búsqueda de todas la entradas *grant* en donde se especifican sus privilegios dentro de todos los ficheros de políticas de seguridad del sistema, la suma de todos los privilegios otorgados en las entradas *grant* darán como resultado el objeto de permisos para dicha autoridad el cual será utilizado para crear un nuevo dominio de protección.

Una vez creado el dominio de protección el agente será incluido en este dominio el cual quedará registrado para la futura incorporación de más agentes que pertenezcan a la misma autoridad. Por cada autoridad se crea un *Codesource* que sirve como identificador de ese dominio de protección. Es importante señalar que los permisos otorgados en los ficheros de políticas de seguridad sirven principalmente para restringir el acceso al sistema de ficheros y el número de conexiones a red.

Cuando el *AgentLoader* posteriormente carga más clases requeridas por el agente estas clases son asociadas al mismo dominio de protección al que fue asociada la clase principal del agente sin importar si el dominio de protección ha sido creado a causa de la llegada de ese agente o si ya existía anteriormente.

Todos lo objetos de los cuales haga instancia el agente internamente tendrán los mismos privilegios de acceso que el propio agente.

18.6 VERIFICACIÓN DE ASIGNACIONES Y CUMPLIMIENTO DE PRIVILEGIOS

Finalmente después de que un agente ha sido asociado a un dominio de protección el gestor de recursos llamado *ResourcesManager* se encarga de verificar el consumo de asignaciones de su autoridad, si ésta se encuentra en su límite y no tiene capacidad para crear al nuevo agente éste no podrá ser ejecutado, en caso contrario el agente ejecutará los métodos correspondientes a su llegada[102] e iniciará su ejecución.

Al mismo tiempo que se inicia el agente se actualizará el registro correspondiente en la tabla *DBResourcesFree* que se encuentra asociado a cada dominio de protección. La gráfica que describe brevemente el proceso de actualización (18.6.1) y que es la continuación de la figura 18.5.1 se presenta en la siguiente página.

La idea de poder asignar una cantidad de recursos especifica a cada autoridad remota y que se vaya actualizando conforme se vayan consumiendo por sus agentes fue introducida a los sistemas basados en Java por MILENIO. Estas asignaciones permiten otorgar recursos cuantitativa y cualitativamente dependiendo del recurso que se trate.

Actualmente los sistemas de agentes móviles basados en Java no ofrecen un mecanismo adicional de seguridad al que ofrece la versión de Java en el que han sido implementados, como la mayoría de los sistemas han sido implementados en el JKD 1.1.6 sólo cuentan con la seguridad que ofrece el *sandbox* la cual tiene muchas carencias y deficiencias. MILENIO además de ofrecer la seguridad de Java2 provee un mecanismo de asignación y verificación de recursos adicional que es llevado a cabo en tiempo de ejecución mediante el uso de asignaciones.

[102] El método típico es *OnArrival()*.

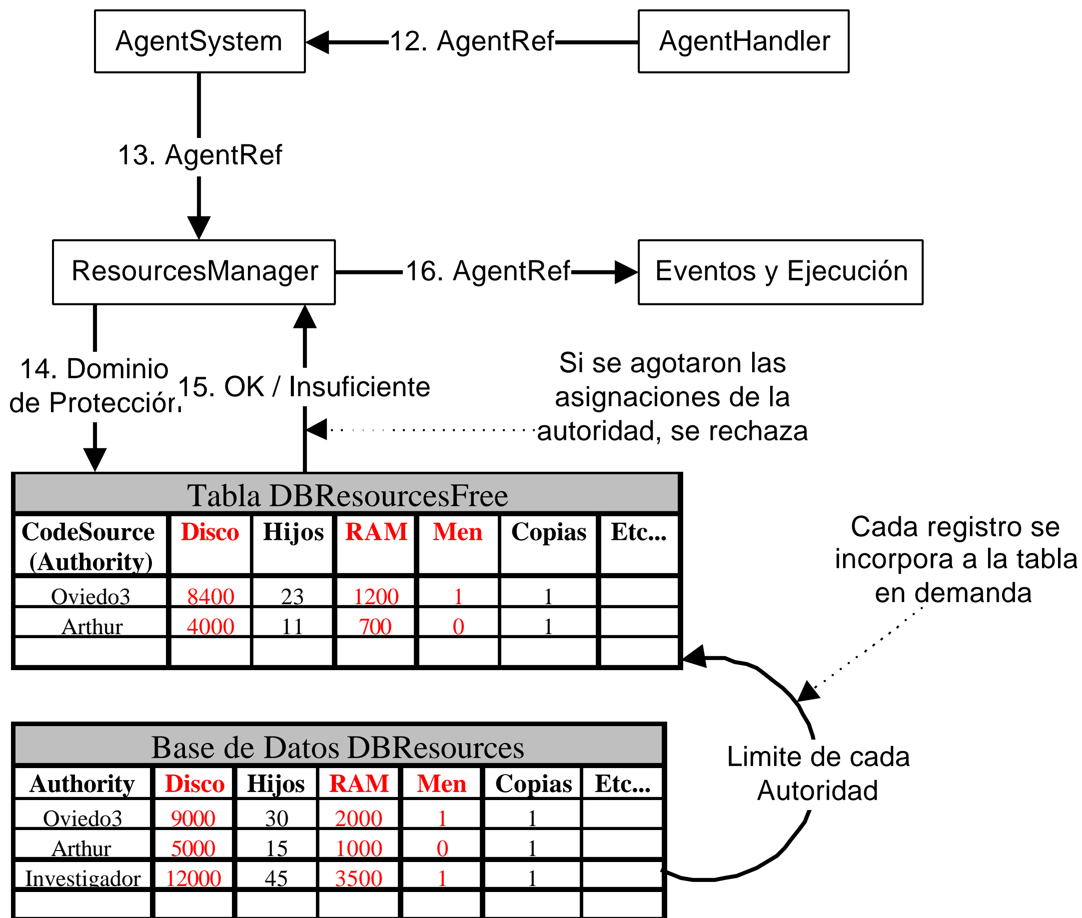


Tabla DBResourcesFree						
CodeSource (Authority)	Disco	Hijos	RAM	Men	Copias	Etc...
Oviedo3	8400	23	1200	1	1	
Arthur	4000	11	700	0	1	

Base de Datos DBResources						
Authority	Disco	Hijos	RAM	Men	Copias	Etc...
Oviedo3	9000	30	2000	1	1	
Arthur	5000	15	1000	0	1	
Investigador	12000	45	3500	1	1	

Figura 18.6.1 – Verificación y Actualización de Asignaciones

18.6.1 DBResourcesFree y DBResources

Cada registro en la tabla *DBResourcesFree* contiene la cantidad de recursos que quedan disponibles para una autoridad específica, es decir la cantidad de recursos de la que pueden disponer los nuevos agentes que lleguen al sistema que pertenezcan a esa autoridad. Cada registro esta identificado por el *CodeSource* que corresponde de forma única a cada autoridad, adicionalmente consta de un objeto *ResourcesAvailable* cuyos campos se muestran individualmente en la tabla.

Cada vez que llega un agente se verifica si su autoridad aún tiene suficientes recursos para satisfacer la asignación solicitada por él, en dicho caso el agente ingresará al sistema y el registro correspondiente a su autoridad en *DBResourcesFree* se actualizará restando a la cantidad actual los valores que requiere el agente, es decir los valores de la asignación individual.

Cuando no existe la autoridad del agente en la tabla *DBResourcesFree* se accede a la base de datos *DBResources* de donde se obtienen los límites máximos especificados para dicha autoridad remota en el sistema, a esos límites máximos se le resta el valor solicitado por la asignación del agente y se da de alta el registro en *DBResourcesFree.*

Es importante señalar que el control de RAM, de espacio en disco y de la capacidad de envíar o no mensajes no han sido implementados(razón por la que se encuentran en un

color más claro) en esta primera versión del sistema. La razón fundamental es que la máquina virtual de Java no provee medios para verificar el consumo de estos recursos en un hilo.

18.6.2 Verificación de las asignaciones individuales del agente

La arquitectura de seguridad además de vigilar que no se agoten los recursos otorgados a cada autoridad también vigila que el agente no agote los recursos que le fueron otorgados a él individualmente es su asignación.

Cada vez que un agente intenta crear un hijo o duplicarse, primeramente se verifica que le haya sido concedida la facultad para hacerlo, en caso afirmativo se crea el nuevo agente siempre y cuando no haya agotado aún el número de agentes que puede generar.

18.6.3 Cumplimiento de privilegios

Finalmente una vez que el agente ha ingresado al sistema de agentes remoto y se le han asignado los privilegios en función de la autoridad a la que pertenece y que se han actualizado las asignaciones disponibles de su autoridad, es necesario que el sistema de seguridad se encargue de verificar que el agente no se exceda en los permisos otorgados, es decir, que se vigile y garantice que sólo podrá hacer en el sistema todo lo que se le ha permitido.

El cumplimiento de privilegios puede considerarse que se lleva a cabo en dos fases, en la primera fase se decide si se le permite o no el acceso al sistema en función de sí tiene privilegios para hacerlo. Es decir, la política de cumplimiento de privilegios se encarga de permitir el acceso sólo aquellos agentes cuyas autoridades puedan ser autentificadas por medio del *keystore* que son las que el administrador del sistema desea que ingresen. Aquellos agentes cuyas autoridades no se encuentren o bien cuya firma digital no sea válida serán rechazados. Véase figura 18.6.3.1.

La segunda fase de supervisión de cumplimiento de privilegios sólo se efectúa sobre los agentes que han logrado entrar al sistema. Esta segunda fase la lleva a cabo el gestor de recursos de SAHARA llamado *ResourcesManager* y el sistema de seguridad de Java apoyado por otros elementos de la arquitectura. La figura 18.6.3.1 muestra de forma general los elementos que intervienen en la supervisión de recursos del sistema.

Cada vez que un agente quiera iniciar una conexión de red, abrir una ventana, acceder al sistema de ficheros o bien realizar una tarea especifica sobre los recursos del sistema. El sistema de seguridad de Java apoyado por elementos de la arquitectura de SAHARA verificará en los dominios de protección del sistema que la autoridad a la que pertenece dicho agente tenga los privilegios apropiados para realizar la acción que pretende.

Si la autoridad goza de dicho permiso en el dominio de protección, la acción solicitada será atendida y el agente podrá hacer lo que deseaba, en caso contrario se lanzará una

excepción y la ejecución de la acción deseada será rechazada evitando con ello un acceso indebido o no autorizado a los recursos del sistema.

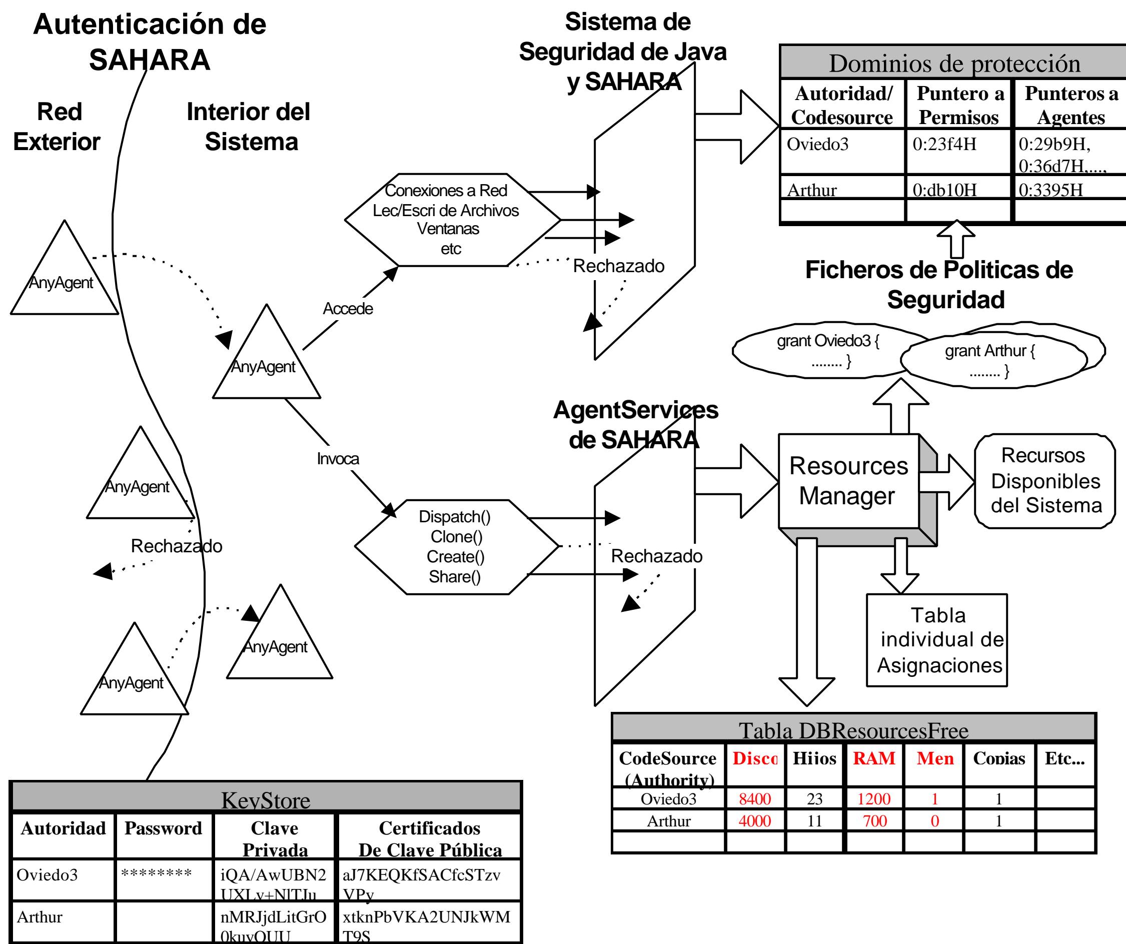


Figura 18.6.3.1 – Supervisión del cumplimiento de privilegios

Por otra parte cada vez que el agente quiera crear un nuevo agente, duplicarse, enviarse a otro sitio o compartir los permisos a uno de sus ficheros, su solicitud pasará a través del proveedor de servicios de agentes de SAHARA quien a su vez solicitara ayuda al gestor de recursos *ResourcesManager* para que verifique si el agente está autorizado para hacer la tarea que solicita.

Cuando llega un agente el gestor de recursos verificará que su autoridad aún tenga asignaciones suficientes en *DBResources* para satisfacer la demanda individual del agente. Posteriormente cuando el agente este en ejecución y solicite al proveedor de servicios de agentes alguna acción, el gestor de recursos verificará que el agente aún cuente con recursos disponibles en su asignación individual. Esto lo verifica en la tabla individual de asignaciones.

Finalmente como veremos en los temas siguientes cuando un agente desea compartir ficheros con algún otro o bien cuando el gestor de recursos detecte que el sistema se encuentra escaso de recursos. El sistema a través del gestor de recursos modificará los ficheros de políticas de seguridad del sistema reduciendo la cantidad de permisos,

inmediantamente después refrescará la política para que los cambios entren en vigor inmediatamente y el sistema permanezca estable.

El mecanismo de cumplimiento de privilegios de MILENIO está integrado al sistema de seguridad lo que favorece su portabilidad hacia otros sistemas de agentes móviles, todas las comprobaciones que se realizan se llevan a cabo por invocaciones del núcleo al sistema de seguridad. No se requiere de ningún preprocesador del interprete como el que se plantea en [HAGI97] lo cual dificulta la incorporación de la arquitectura de seguridad a otros trabajos ni tampoco se necesita un interprete seguro como *safe Tcl* que es utilizado por *D'Agents.*

La principal desventaja de la utilización del interprete seguro es el bajo tiempo de respuesta de la aplicación así como la poco posibilidad de incorporar un modelo de seguridad basado en un preprocesador o en un interprete seguro a otros sistemas de agentes móviles basados en Java.

18.7 PERMISOS OTORGADOS POR AGENTES

En ciertas aplicaciones se requiere del trabajo en paralelo y en grupo de varios agentes. Para ello es necesario que un agente sea capaz de crear ficheros que luego sean accedidos por otros agentes y no necesariamente de su misma autoridad, por lo que es indispensable que un agente pueda otorgar privilegios específicos sobre un fichero a los agentes de otras autoridades.

Para permitir esto, cada agente que adquiere su funcionalidad heredando de la clase genérica *Agent*, heredara de esta clase un método llamado *share* que le permitirá solicitar a la clase proveedora de servicios de agentes la inclusión de un fichero o directorio en el archivo que define la política de seguridad del sistema.

Como se describió en el capítulo XII, el método *share* permite especificar además del fichero que será incluido, las autoridades y los privilegios que le serán otorgados en el fichero de políticas de seguridad.

Cuando un agente invoca el método *share*, la clase proveedora de servicios de agentes invoca al gestor de recursos para que verifique si el agente está autorizado para realizar esta tarea, de ser así, el gestor de permisos *PermissionManager* realiza los cambios necesarios en el fichero de políticas de seguridad del sistema y enseguida invoca el método *refresh()* que se encarga de actualizar la política de seguridad en vigor.

De esta manera los nuevos agentes que lleguen al sistema estarán sujetos a una nueva política de seguridad, los nuevos agentes que pertenezcan a una de las autoridades a las que se les hayan dado nuevos permisos, podrán efectuar accesos a los ficheros incluidos de acuerdo a los permisos que se le hayan otorgado a su autoridad ya que serán asociados a un nuevo dominio de protección en donde estarán incluidos los privilegios otorgados mediante el método *share.*

Esta facilidad de que los agentes puedan proteger y otorgar privilegios sobre los ficheros que generan especificando con quien los comparten y los privilegios que les

otorgan es una característica exclusiva de MILENIO, los sistemas actuales para poder modificar la política de seguridad tienen que detener el sistema, cambiar la política y luego volver arrancar el sistema, MILENIO logra cambiar la política de seguridad en tiempo de ejecución sin tener que detener el sistema.

18.8 AUTOCONTROL DEL SISTEMA Y MODIFICACIÓN DE LA POLÍTICA DE SEGURIDAD EN TIEMPO DE EJECUCIÓN

Esta característica descrita en capítulo XV se implementa parcialmente en esta versión de MILENIO ya que sólo se verificará por el consumo de memoria RAM, sin embargo el diseño del sistema esta preparado para incorporar fácilmente el resto de la funcionalidad que SAHARA ofrece en este punto.

El autocontrol de sistema es una tarea que llevan a cabo en paralelo el gestor de recursos y el gestor de permisos del sistema. La finalidad de esta tarea es mantener al sistema de forma estable y que no tenga un desequilibrio debido a la escasez de los recursos básicos. El gestor de recursos verifica la escasez de recursos mientras que el gestor de permisos restringe los permisos apropiados para evitar que la escasez continúe.

Para lograrlo el gestor de recursos monitoriza periódicamente mediante la función del sistema *freeMemory()* la cantidad de RAM, cuando detecta escasez del recurso de acuerdo a las cuotas máximas y mínimas establecidas entonces impide la creación o recepción de más agentes en el sistema para evitar que se siga consumiendo memoria.

En versiones futuras también se verificará el espacio en disco duro y las conexiones a red, en este caso cuando se detecte alguna escasez se llamará al gestor de permisos para que edite y modifique los ficheros de políticas de seguridad restringiendo los permisos de escritura o de apertura de nuevas conexiones de red, posteriormente al igual que con el método *share* el gestor de permisos tendrá que invocar el método *refresh()* para actualizar la política de seguridad y surtan efecto los cambios.

Las verificaciones anteriores al igual que el consumo de RAM por agente serán incluidas en futuras versiones de MILENIO cuando la máquina virtual de Java ofrezca funciones para saber el consumo de memoria y de disco duro por cada hilo de ejecución en el sistema.

No obstante esta característica de autosupervisión y autocontrol del sistema al igual que la anterior sólo está presente en MILENIO, el autocontrol brinda una confianza adicional sobre la estabilidad del sistema y de que no se vendrá abajo por escasez de recursos. El resto de los sistemas actuales basados en Java no cuentan con dicha funcionalidad, sólo tienen mecanismos de seguridad para protegerse de los agentes pero no monitorizan los recursos disponibles para poder prever una futura escasez y evitar una posible corrupción del sistema.

18.9 LO QUE NO SE IMPLEMENTO DE SAHARA EN MILENIO

En esta primera versión de MILENIO por diversas razones no se implemento la arquitectura de SAHARA en 100%, a continuación indico cuales han sido las carencias de esta primera versión que serán implementadas en las versiones venideras.

1. La gestión de las asignaciones para controlar el consumo de memoria RAM y espacio en disco duro no se implemento ya que el lenguaje Java no ofrece ningún método o interfaz que permita conocer el consumo por hilo de estos elementos. En cuanto la JVM ofrezca un mecanismo para hacerlo o encontremos un medio fruto de nuestras investigaciones, dicha gestión se incluirá inmediatamente.
2. La capacidad de habilitar o inhabilitar el envío o recepción de mensajes mediante las asignaciones no fue incluido debido a que la versión de MILENIO 1.0 no tiene soporte para comunicación entre agentes mediante mensajes síncronos o asíncronos.
3. No fue incluido ninguno de los mecanismos que ofrece SAHARA para proteger el estado de datos de los agentes (por ejemplo los PRAC’s) debido a su complejidad de implementación. Éstos serán incluidos en versiones futuras.
4. El concepto de lugares para crear grupos de agentes que compartieran recursos y ficheros fue implementado parcialmente, ya que se implemento el método *share* pero no se realiza a fondo las gestiones de grupos.
5. El concepto de regiones y grupos de regiones para crear redes confiables será implementado en la siguiente versión cuando sea posible personalizar más fácilmente el sistema y el administrador pueda configurar y modificar las regiones en las que se desee incluir.
6. El autocontrol del sistema que permite modificar la política de seguridad en tiempo de ejecución cuando se detecta una escasez de recursos fue implementada parcialmente ya que sólo se verifica la cantidad total de memoria RAM que queda disponible y cuando se detecta una escasez se modifica la política. El autocontrol del disco duro y el uso de cuotas no se implemento por lo mismo que se indicó en el punto 1.
7. Como SAHARA soporta autenticación múltiple la próxima versión de MILENIO incluirá la posibilidad de firmar agentes por múltiples autoridades.

Capítulo IXX
EL ENTORNO INTEGRADO DE DESARROLLO DE MILENIO

El sistema de agentes MILENIO cuenta con una interfaz gráfica que proporciona un entorno de desarrollo integrado que facilita la implementación de aplicaciones con agentes móviles.

El entorno de desarrollo restringe el acceso sólo a los usuarios autorizados del sistema, permite la creación y monitorización de agentes, la gestión de usuarios y recursos del sistema así como el control de bitácoras de los sucesos que ocurren durante la ejecución del sistema. Finalmente proporciona una ayuda en línea completa sobre las tareas del sistema. En este capítulo se describirán sólo los aspectos relacionados con la creación de aplicaciones. Para una información detallada de todo el sistema refiérase al apéndice A (Manual de usuario).

19.1 ACCESO AL SISTEMA

Al arrancar el sistema, se pide un nombre de usuario y su contraseña para evitar a los usuarios no deseados. Cada usuario representa la autoridad autorizada para crear, firmar y enviar agentes a servidores remotos, por lo que en realidad una cuenta puede ser compartida por más de una persona. El administrador es el único usuario que puede dar de alta en el sistema a nuevas autoridades/usuarios.

- *Nombre de usuario:* representa la autoridad con la que se firmarán y se enviarán los agentes que se creen en el sistema.
- *Contraseña:* clave de acceso al sistema.

La identidad del usuario así como su contraseña se comprueban en el KeyStore. La ventana de acceso se aprecia en la figura 19.1.1.

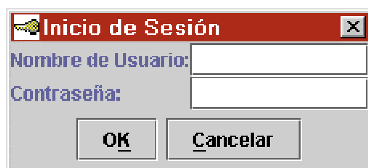


Figura 19.1.1 Ventana de Acceso

19.2 VENTANA PRINCIPAL DEL SISTEMA

Una vez iniciado el sistema se accede a la ventana principal, que es la ventana central de la interfaz desde donde se puede acceder a todas las opciones del sistema. La ventana se puede dividir en 5 zonas:

1. Barra de menús,
2. Barra de herramientas,
3. Área de agentes activos,
4. Salida estándar y de error,
5. Barra de estado.

Dentro de la barra de menús algunas opciones no estarán activas (permitidas) para los usuarios normales sino sólo para el administrador. La ventana principal se aprecia en la siguiente figura:

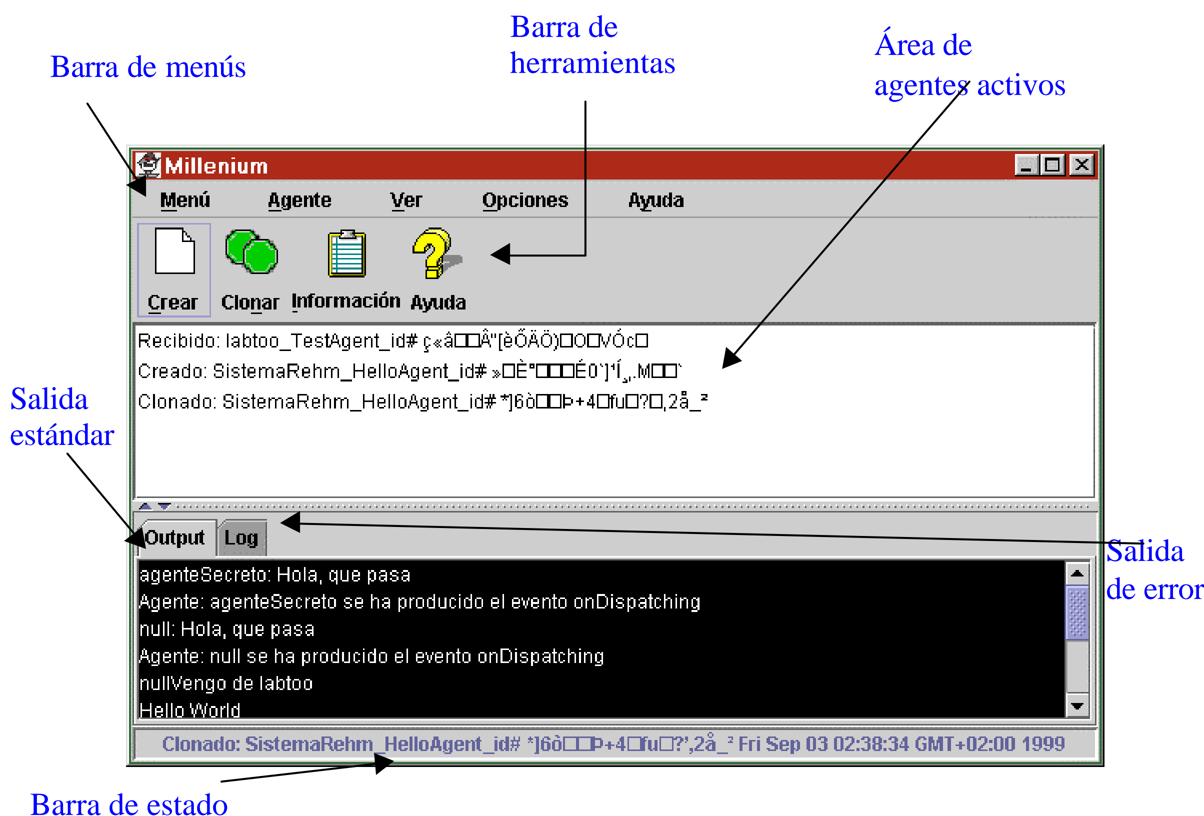


Figura 19.2.1 Ventana Principal de Milenio

19.2.1 La barra de Menús

En ella se encuentran un enlace a todas las acciones a las que se puede acceder desde la interfaz gráfica. Se encuentran agrupadas en: Menú, agentes, ver, opciones y ayuda. Cada menú presenta a su vez sus respectivas opciones conteniendo submenús algunas de ellas.

19.2.2 La barra de herramientas

Permite realizar las principales tareas sobre los agentes: creación, clonación, visualización de información sobre agente y el acceso a la ayuda. Cuando el usuario sobrepasa con el puntero del ratón sobre un botón, éste se resaltará e indicará mediante una etiqueta la acción que se ejecutará si se pulsa el botón. La barra de herramientas puede ocultarse o mostrarse a través del menú ver.

19.2.3 Area de Agentes Activos

Esta zona muestra los agentes que se encuentran activos en el sistema, es decir los agentes que se encuentran en ejecución. Cada vez que un agente es dado de alta en el sistema, bien mediante una creación, una clonación o mediante la llegada de un agente de un sistema remoto, la tabla de agentes activos es actualizada automáticamente por el sistema. Estas modificaciones se reflejan de forma inmediata en la interfaz. De igual forma, cuando un agente abandona el sistema, bien porque haya concluido su ejecución o porque migre a otro sistema, la tabla de agentes activos y la interfaz se actualizan.

Sobre esta zona se puede desplegar un menú pop-up (pulsando el botón derecho del ratón), que permite acceder a ciertas operaciones de manera más rápida. Para que el menú pop-up aparezca, deberá seleccionarse previamente una línea de la lista. Esta línea representa el agente sobre el que se realizará la operación elegida.

Cada línea de la ventana sigue el siguiente formato:

<*NombreSistema*>_<*NombreClase*>_id#<*Identificador*>

Separando la ventana de agentes activos de las ventanas de salidas se encuentra una línea divisoria que permite la modificación del tamaño de las mismas

19.2.4 Ventanas de salida estándar y de error

El objetivo de la ventana de salida estándar es visualizar los mensajes de salida que envían los agentes mediante llamadas a métodos de la clase System.out.

También se recoge en la pestaña log la salida de error, en la que se muestran tanto mensajes del sistema como mensajes de error(excepciones). El sistema muestra mensajes que permiten seguir los pasos que se van realizando durante el ciclo de vida de los agentes así como los pasos que lleva a cabo cuando envía agentes a otro servidor. Ambas ventanas disponen de un menú pop-up que permite borrar el área de texto.

19.2.5 la barra de estado

Se encuentra situada en la parte inferior de la ventana y muestra la última operación realizada por el sistema de agentes.

19.3 MENÚ AGENTE

La barra de menús, dispone de un menú Agente desde el cual se pueden realizar las principales acciones sobre los agentes.

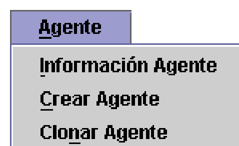


Figura 19.3.1 Menú Agente

19.3.1 Crear nuevo Agente

Existen tres mecanismos para desplegar la ventana de creación de un nuevo agente: mediante la opción del menú, mediante el botón de la barra de herramientas y mediante un menú desplegable(pop-up) que aparece al pulsar el botón derecho del ratón sobre la zona de agentes activos del menú principal.

El cuadro de diálogo que permite la creación de un nuevo agente en el sistema es el siguiente.

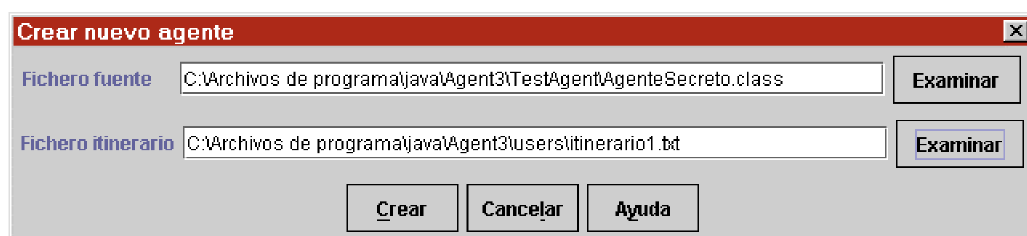


Figura 19.3.1.1 Ventana Crear Agente

19.3.1.1 Seleccionar Clase

Es imprescindible que se especifique la ruta completa en la que se encuentra el código del agente(fichero con extensión "*.class*"). Esta clase será una clase programada por el usuario. Para que la clase pueda ejecutarse deberá haber heredado de la clase *Agent* del sistema. La clase *Agent* es una clase abstracta, es decir, no se pueden crear instancias de ella directamente. El programador de agentes será el encargado de programar las clases a partir del API[103] del sistema.

La ruta del fichero puede introducirse manualmente o bien pulsando sobre el botón Examinar, que despliega una ventana que permite navegar sobre el sistema de ficheros.

Al pulsar el botón Examinar se despliega un cuadro de diálogo como el siguiente:

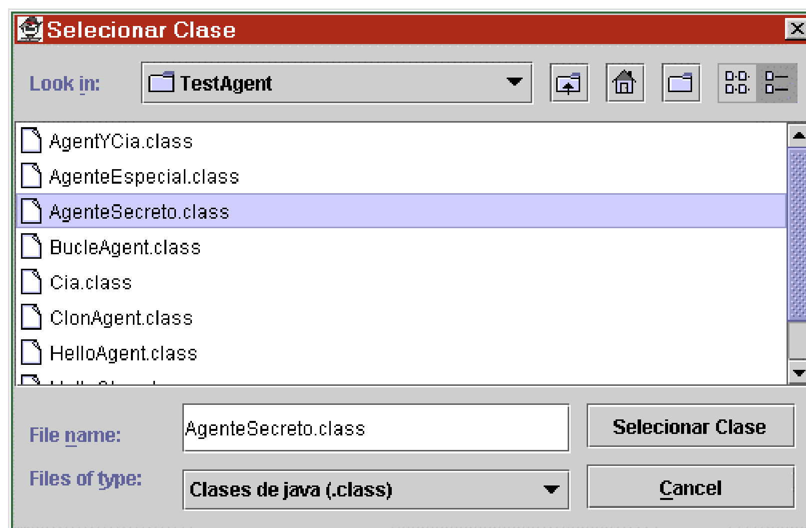


Figura 19.3.1.1.1 Seleccionar clase Agente

El directorio sobre el que aparece este diálogo corresponde al directorio home de autoridad que este haciendo uso del sistema.

19.3.1.2 Seleccionar Itinerario

El agente podrá llevar consigo un itinerario que determine el grupo de servidores destino por los cuales desea viajar, procesar la tarea señalada en cada servidor, recopilar la información deseada, y posteriormente retornar a su lugar de origen con los datos. Dicho itinerario podrá ser modificado dinámicamente en cualquier punto de su recorrido.

El itinerario es un campo opcional en algunos agentes, es decir, no es necesario especificar itinerario alguno para su ejecución. También puede ocurrir que aunque se especifique un itinerario, el agente no haga uso del mismo. ¿Para qué sirve entonces el itinerario? Un agente dispone de dos métodos para migrar a otro sistema:

[103] API: Application Programming Interface.

1- *dispatch(java.net.URL)* Ejemplo: *dipatch(new URL("156.35.82.3:1975"))*
2- *dispatch(int)* Ejemplo: *dispatch(0)*

En el primer caso, el agente indica el sistema al que quiere acceder mediante su dirección y puerto, mientras que en el segundo se indica la dirección a través de un índice del itinerario(en este caso la posición 0 del itinerario, es decir la primera línea del fichero).

En el segundo caso, debe de establecerse un itinerario para el agente, en caso contrario se producirá una *excepción*. La ventaja de utilizar itinerarios es poder cambiar la ruta de un agente sin tener que volver a compilarlo, basta con modificar el fichero que contiene el itinerario, de igual forma facilita que el propio usuario en función de la información que vaya recopilando pueda cambiar su rumbo hacia otro sitio.

La forma de especificar el itinerario es introduciendo el nombre del fichero que lo contiene. El usuario que desee establecer un itinerario deberá especificar la ruta completa del fichero en el que se encuentra. El fichero del itinerario no es más que un fichero de texto normal. El formato del fichero es muy simple pero estricto: se indica en cada línea la dirección del sistema junto con el puerto, separados por el carácter ':'. Un ejemplo válido de fichero itinerario[104] es el siguiente:

```
156.35.82.3:1975
156.35.82.2:1973
156.35.82.4:1974
```

Figura 8.3.3 Fichero de Itinerario

Una vez que el agente se crea con su itinerario, éste se almacena en una estructura de datos del agente y pasa a formar parte de su estado de datos, de forma que cuando el agente migra a otro sistema, se lleva el itinerario consigo.

19.3.2 Clonar Agente

El usuario puede clonar cualquier agente que se encuentre en el sistema que le pertenezca, tanto si el agente es local como si es remoto. La única condición que debe cumplirse es que la autoridad tenga recursos disponibles para clonarse.

Para clonar un agente basta con seleccionarlo de la lista de agentes activos(ver ventana principal) y seleccionar en el menú Agente la opción Clonar Agente. También se puede clonar el agente pulsando el botón correspondiente de la barra de herramientas o desplegando el menú pop-up sobre la zona de agentes activos(ver ventana principal).

[104] Nótete que no se debe de indicar ningún tipo de protocolo ("http://",...).

Un agente puede clonarse también sin la intervención del usuario mediante la llamada al método *clone()* del agente.

Cuando el sistema realiza un clone de un agente lo que hace en realidad es crear otra instancia de la clase que le dio origen con los mismos permisos y atributos, por supuesto la única diferencia es que le asigna un identificador distinto.

19.3.3 Información de agente

El usuario puede obtener información a cerca de un agente que se encuentre en el sistema mediantre tres vías: mediante la opción del menú, mediante el botón de la barra de herramientas y mediante un menú desplegable(pop-up) que aparece al pulsar el botón derecho del ratón sobre la zona de agentes activos del menú principal.

La información que se muestra para el agente es la siguiente:

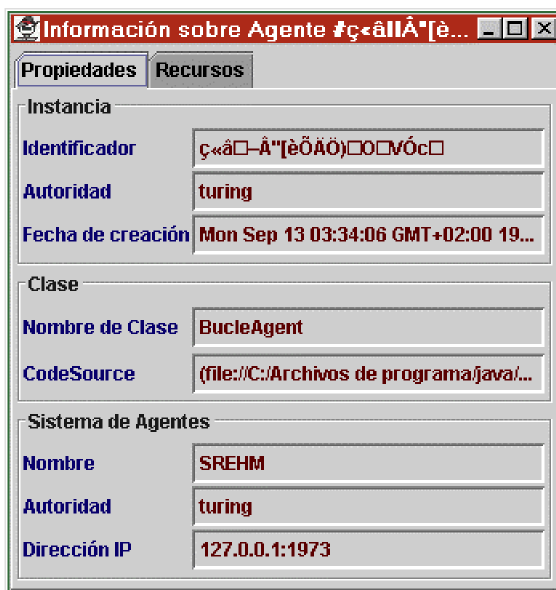


Figura 19.3.3.1 Información Agente: Propiedades

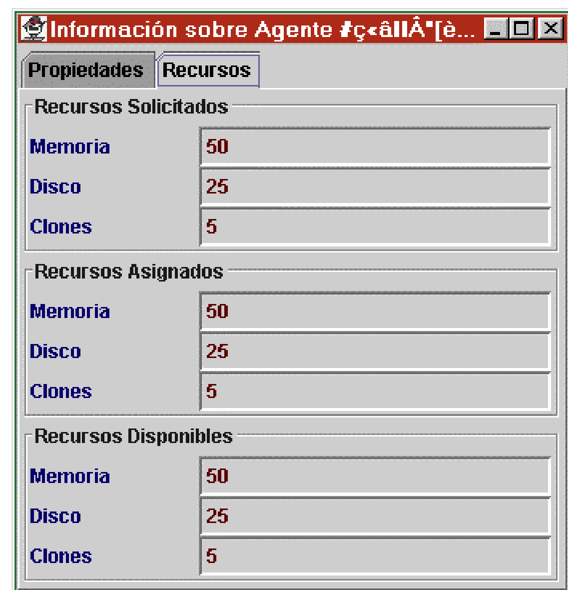


Figura 19.3.3..2 Información Agente: Recursos

La ventana se compone de dos paneles:

1. En el primer panel se muestran las propiedades del agente. Este panel se divide a su vez en tres zonas: datos sobre la instancia, datos sobre la clase a la que pertenece el agente indicando su *codesource* que representa el directorio que se le ha asignado a su autoridad y al cual se le otorgan los permisos en el fichero de políticas de seguridad, y finalmente se muestran los datos sobre el sistema de origen del agente.

2. El segundo panel muestra información sobre los recursos o asignaciones del sistema. El panel también se divide en tres zonas: recursos solicitados, asignados y disponibles.
   - *Recursos solicitados:* son las preferencias sobre asignaciones solicitadas por el agente al sistema. Cuando el agente se crea por primera vez en su sistema de origen, se crea un objeto en el que especifica los recursos que requiere para su ejecución. Estos datos son leídos del fichero */config/DBPreferences* y

han sido establecidos por el administrador en el momento de dar de alta a la autoridad a la que pertenece el agente.

- *Recursos asignados:* son los recursos asignados por el sistema. Si el agente es local, se le asignan tantos recursos como solicite. Si el agente es remoto, cuando el agente llega del sistema remoto, se comprueba que para la autoridad a la que pertenece queden asignaciones disponibles en el sistema. Si para dicha autoridad quedan asignaciones, se le asignan al agente y éste comenzará a ejecutarse. En caso contrario, si no quedan asignaciones suficientes, bien porque haya otros agentes en el sistema de la misma autoridad que estén consumiendo recursos o bien porque el agente solicite más recursos de los que dicha autoridad tenga asignados, se le asignarán los recursos que queden disponibles hasta completar el máximo permitido para dicha autoridad.
- *Recursos disponibles:* son los recursos, que el agente aún no ha consumido de los que le han sido asignados.

19.4 MENÚ VER

El usuario dispone del menú Ver con las siguientes opciones:

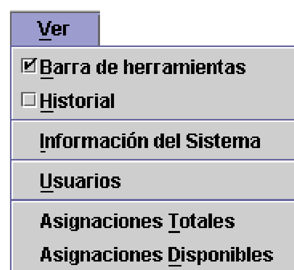


Figura 19.4.1 Menú Ver

19.4.1 Barra de herramientas

Desde el menú Ver se puede mostrar u ocultar la barra de herramientas en la ventana principal. Por defecto la barra aparece visible.

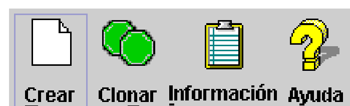


Figura 19.4.2 Barra de herramientas

Desde esta barra se pueden realizar las operaciones: crear, clonar y mostrar información de agente. También dispone de un botón que despliega la ventana de ayuda.

19.4.2 Historial

La interfaz muestra una lista en la que se muestren todos los agentes activos en el sistema: área de agentes activos en la ventana principal. Cada vez que un agente sea dado de alta en el sistema, está lista deberá actualizarse, indicando el nombre del sistema del agente, su clase y su identificador. De igual forma, cuando un agente abandona el sistema o muere, será eliminado de dicha lista. No obstante, la interfaz dispone de un histórico en el que se visualizará el proceso de altas y bajas en el sistema.

Desde el menu Ver se puede visualizar el historial de las operaciones realizadas en el sistema.

Cada línea de la ventana del historial tiene el siguiente formato:

<*Acción*>: <*NombreSistema*>_<*NombreClase*>_id#<*Identificador*> <*fecha*>

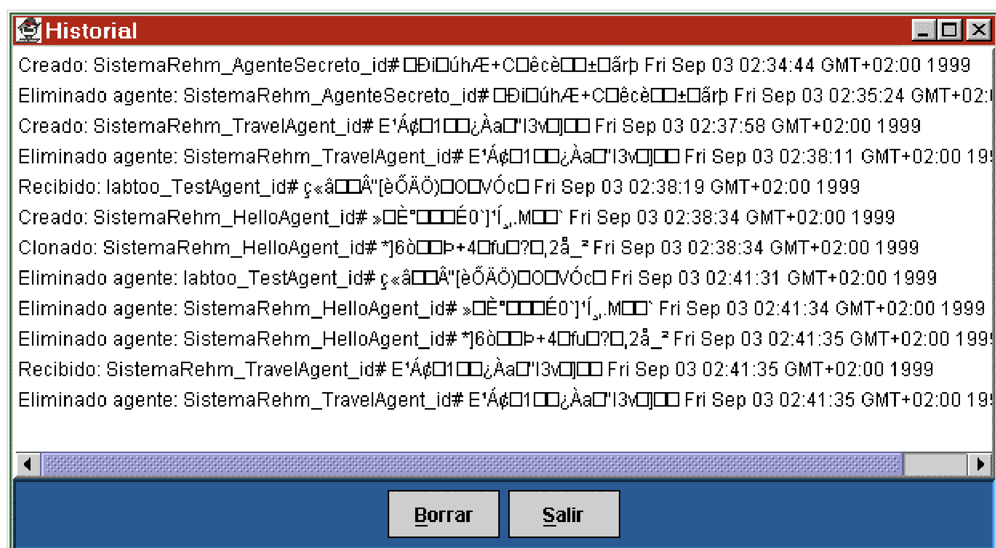


Figura 19.4.2.1 Ventana Historial del Sistema

Para evitar que la ventana de historial crezca de una forma ilimitada, cada vez que se llega a cien líneas, la venta se limpia guardando los datos del historial en un fichero de bitácora del sistema.

19.4.3 Información del Sistema

Esta opción despliega una ventana que muestra información sobre el sistema. En la zona superior se muestran los parámetros del sistema y la **autoridad** o usuario del mismo. Los parámetros del sistema son el **nombre** común, la **dirección IP** y el **puerto**. Cuando el sistema arranca, toma unos valores por defecto para el nombre y el puerto(que serán establecidos durante el proceso de instalación) y la dirección IP se tomará la asignada a dicho ordenador. Sin embargo, estos parámetros podrán ser modificados por el usuario en momento de arranque del sistema. Una vez iniciado el sistema, estos parámetros no podrán ser modificados. El puerto debe ser un número entero comprendido entre 1.024 y 65.535.

Figura 19.4.3.1 Ventana Información del Sistema

La autoridad es el usuario que arrancó el sistema. El administrador es el único usuario que tiene capacidad de dar de alta a nuevos usuarios.

En la zona inferior se muestra la cantidad de memoria total y disponible del sistema. La cantidad de memoria total puede aumentar.

El usuario puede invocar a través de la interfaz al recolector de basura(*garbage colector*) de forma que se elimine la memoria ocupada por los objetos que ya no son accesibles. Si el usuario no la invoca manualmente la máquina virtual de Java lo hará periódicamente.

19.4.4 Usuarios

La ventana de administración de usuarios muestra la lista de usuarios que hay en el sistema. Se muestra para cada usuario si es local o remoto y sus asignaciones(disco, memoria y clones).

Administrador de Usuarios

Usuarios

Usuario	Certificados	Tipo	Disco/Memoria/Clones/canClone/mess...
arthur	a - a - a - a - a - a	Remo...	25 - 50 - 7
administrador	admin - admini - admin - admin - admin - es	Local	5 - 6 - 7 - true - false
turing	turing - t - t - t - t - t	Local	25 - 50 - 5 - true - true
renato	r - r - r - r - r - r	Local	5 - 5 - 1000 - true - true

Figura 19.4.4.1 Lista de usuarios

Los usuarios/autoridades locales son los que podrán acceder al sistema mientras que los usuarios/autoridades remotos sólo podrán enviar agentes a dicho sistema. Cuando el usuario es local, las asignaciones representan los recursos que van a solicitar los agentes al ser dados de alta en el sistema.

Cuando la autoridad es remota, estas asignaciones representan los recursos máximos que pueden ser asignados en el sistema a todos los agentes remotos que provengan de la misma autoridad. Es decir, la suma de los recursos asignados a los agentes de una misma autoridad, nunca puede superar este máximo.

Cuando el usuario del sistema es el administrador, la interfaz le permite desplegar el siguiente menú contextual pulsando sobre el usuario deseado con el botón derecho del ratón.

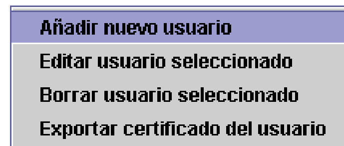


Figura 19.4.4.2 Menú Pop-up de Usuarios

Dicho menú le permitirá editar el contenido de la autoridad seleccionada, borrarla o bien exportar su certificado.

19.4.5 Asignaciones totales

Esta ventana muestra las asignaciones máximas permitidas para cada **autoridad remota**. Estas cantidades son establecidas por el administrador del sistema en el momento en el que dio de alta a cada autoridad remota. Los datos de esta ventana son fijos mientras el administrador no los modifique. (Véase en el capítulo XVIII la figura 18.1.2).

Asignaciones Totales para cada Autoridad

CodeSource	Memoria	Disco	Clones
http://156.35.82.4:1974/ -> serafin	15	15	15
http://156.35.82.3:1975/ -> arthur	50	25	7

Figura 8.9.7 Asignaciones Totales

19.4.6 Asignaciones disponibles

Son los recursos que aún no han sido asignados de las asignaciones totales. Es decir, son los recursos que quedan aún disponibles para los agentes de cada autoridad. Estos recursos disponibles son dinámicos, van disminuyendo o aumentando según se vayan asignando recursos a los agentes que vayan llegando o bien que éstos los vayan liberando al terminar su ejecución o ir marchando.

19.5 MENÚ AYUDA

El usuario dispone de una ayuda en línea desde la cual puede ver el manual de usuario, la API del sistema y ejemplos de código de agentes, así como otras utilidades que le pueden servir en el desarrollo de aplicaciones como lo muestra la figura 19.5.1.

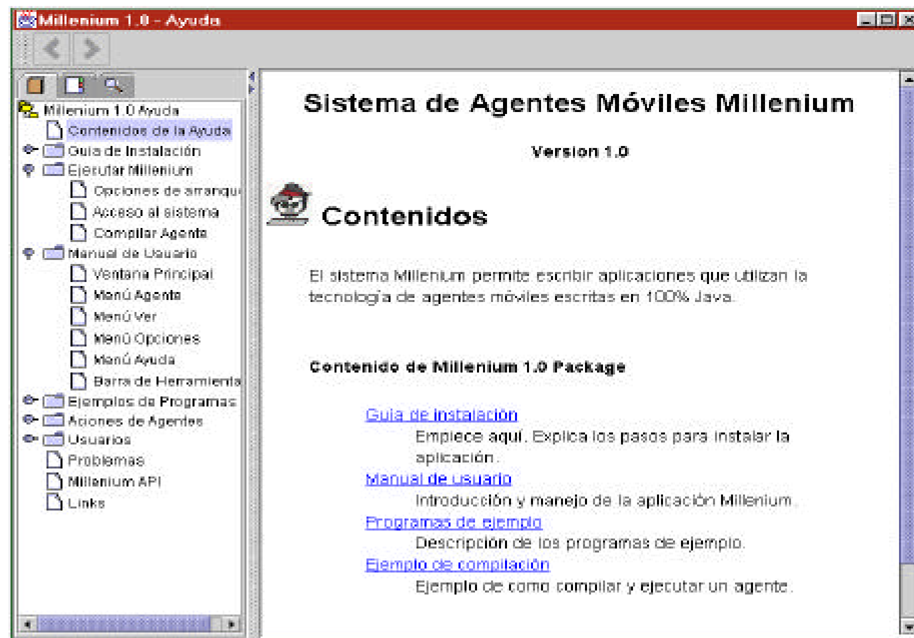


Figura 19.5.1 Ventana Principal de Ayuda

La ayuda se muestra utilizando la utilidad Java Help 1.0. Esta utilidad despliega una ventana en la que se muestra una tabla de contenidos y permite la búsqueda de palabras o frases en todas las páginas de la ayuda, este permite que se busquen todas las páginas en donde se encuentre una frase determinada, por ejemplo “crear agente”, indicando además el porcentaje de coincidencia dentro de la página. (figura 19.5.2).

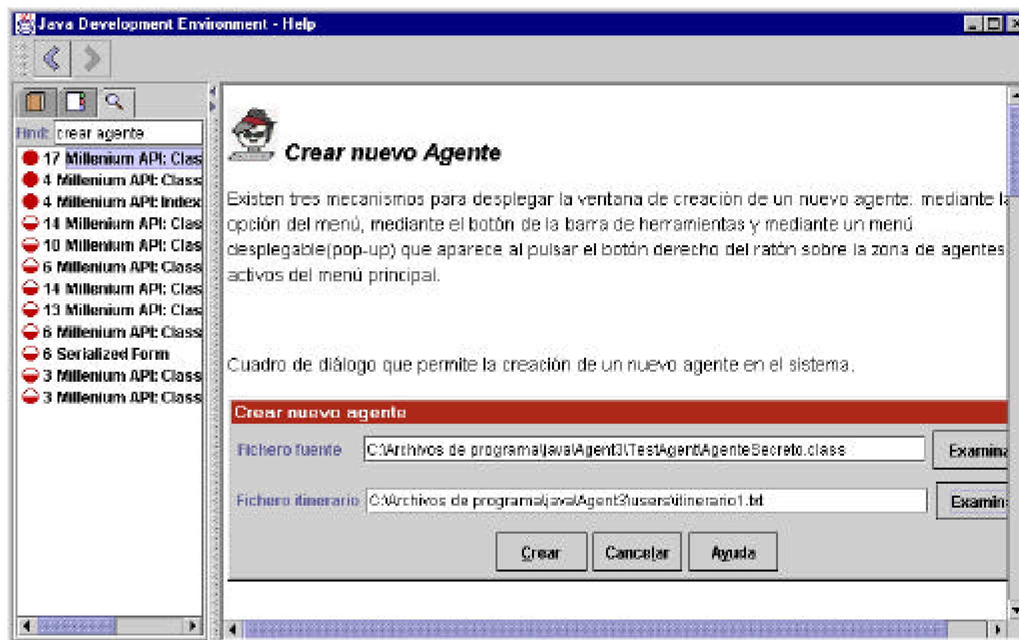


Figura 19.5.2 – Búsqueda de frases o palabras en la ayuda.

El formato de los documentos es HTML estándar, por lo tanto puede utilizarse cualquier otro navegador para visualizarla, no necesitando que el sistema esté en ejecución para consultarla.

PARTE V .- EVALUACIÓN DEL PROTOTIPO, CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS.

Capítulo XX
APLICACIÓN DE SAHARA A OTROS SISTEMAS DE AGENTES MÓVILES Y TRABAJO RELACIONADO

Mientras la comunidad mundial aún no acepta completamente la tecnología de agentes móviles y espera la *"killer application"* que de inicio a una nueva generación de aplicaciones de software, la investigación en este campo se enfoca principalmente en crear mecanismos de seguridad para que los sistemas de agentes móviles superen uno de los principales inconvenientes que tienen actualmente.

La seguridad es uno de los factores más importantes en un sistema de agentes por lo que todos los sistemas actuales gozan de un mecanismo de defensa. Algunos de ellos tienen mecanismos primitivos y poco eficientes mientras que otros ofrecen mecanismos más robustos.

Aunque se podrían describir los sistemas de seguridad de varios sistemas de agentes como Concordia, Voyager, D'Agents, etc., sólo mencionaría mecanismos de protección muy elementales que se reducen en su mayor parte al uso de criptografía para la autenticación de las autoridades de los agentes. Por esta razón es preferible describir los trabajos de investigación más actuales relacionados con la seguridad de sistemas de agentes móviles aunque su implementación no exista o sea prácticamente imposible[105].

A continuación describiré un modelo de seguridad para los aglets[106] creado por Günter Karjoth y Danny Lange creador de los aglets [KARJ97] en él tratan de ofrecer una amplia protección al sistema de agentes. De igual forma se describirá brevemente el trabajo de Fritz Hohl [HOHL97] quien trata de ofrecer una solución al problema más difícil de resolver en sistemas de agentes móviles, es decir, brindar protección a los agentes móviles contra ataques de servidores maliciosos[107]. Finalmente se mencionará como puede influir SAHARA en otros sistemas de agentes móviles actuales.

[105] Esto es debido a que algunos de los trabajos que se presentarán poseen un contenido teórico muy fuerte cuya implementación práctica es casi imposible.

[106] Aunque este no ha sido implementado actualmente con lo que se desconoce su factibilidad y su eficiencia.

[107] Existe otro trabajo de Tomas Sander [SAND97b] para resolver el mismo problema que se basa en la ejecución de programas encriptados, no lo describiré aquí por no tener suficiente relación con SAHARA.

20.1 EL MODELO DE SEGURIDAD PARA LOS AGLETS

Günter desarrolló un modelo de seguridad que proporciona un marco de trabajo general para la seguridad de los aglets. El modelo soporta la definición flexible de varias políticas de seguridad y describe como y donde un sistema seguro refuerza estas políticas.

Las políticas de seguridad se definen en términos de un conjunto de reglas por una autoridad administrativa. Las políticas especifican las condiciones bajo las cuales los aglets pueden acceder a objetos, la autenticación requerida de usuarios y el grado de contabilidad para cada actividad de seguridad relevante.

La concesión de permisos se basa en la autoridad[108] a la que pertenecen los agentes, el modelo de seguridad reconoce las siguientes autoridades:

- *Aglet:* instancia del programa del agente mismo.
- *Productor del Aglet:* autor del programa del agente (humano o compañía).
- *Propietario del Aglet:* individuo que envió el agente.
- *Contexto:* interprete que ejecuta a los agentes.
- *Productor del Contexto:* autor del contexto.
- *Administrador del Contexto:* propietario u operador del contexto.
- *Dominio:* un grupo de contextos poseídos por la misma autoridad.
- *Autoridad del Dominio:* propietario o administrador del dominio.

La jerarquía de autoridades que da prioridad a los permisos otorgados por una entidad particular es:

> *Autoridad del Dominio > Administrador del Contexto >*
> *Propietario del Aglet > Productor del Aglet*

Una base de datos de políticas de seguridad representa la política definida por el Administrador del contexto para proteger los recursos del servidor, mientras que las preferencias de seguridad representan la política definida por el propietario del agente para restringir las actividades que sus agentes pueden hacer.

Tanto el administrador del contexto como el propietario del agente tienen sus propios intereses específicos referentes a lo que un aglet debería ser capaz de hacer, ambos pueden desear restringir sus capacidades. Tales restricciones pueden ser para acceder a los recursos locales de un contexto o al ofrecer servicios a otros aglets.

Este modelo simplifica la administración de las políticas introduciendo la noción de roles, es decir, productor, propietario, etc. Para lograrlo es necesario la creación de un lenguaje que permita definir las políticas y que especifique los intereses de un administrador del contexto y de un propietario de un aglet.

[108] El término principal es usado como sinónimo del termino autoridad dentro del modelo de seguridad de los aglets, por lo que puede encontrarse indistintamente en documentos que hablan sobres seguridad de los aglets

Adicionalmente al lenguaje es necesario especificar los privilegios que podrán ser otorgados así como el formato de la base de datos de políticas del administrador del contexto y de las preferencias del propietario del agente.

20.1.1 El lenguaje de autorización

El lenguaje soporta grupos, autoridades compuestas (un conjunto de autoridades), y una jerarquía de recursos con permisos asociados. Para permitir un control individual, una política de seguridad consiste de un conjunto de privilegios nombrados y de las autoridades que se hacen corresponder con los privilegios. Más aún el lenguaje permite la definición de listas negras que deshabilitan a los aglets y a contextos conocidos de la ejecución de ciertas tareas.

Inicialmente el lenguaje permite definir autoridades simples y compuestas. Algunos ejemplos de autoridades simples son:

- Aglet=MyAglet – denota al aglet MyAglet.
- Owner=Arturo – denota a los agentes enviados por Arturo.
- Master=Apolo – denota a los agentes que llegan directamente de contexto administrados por Apolo.

Por su parte las autoridades compuestas ofrecen una forma conveniente de combinar privilegios que deben ser concedidos a varias autoridades en un permiso de acceso individual.

Por ejemplo la siguiente especificación une dos productores que gozaran de un mismo privilegio dentro de un contexto determinado.

GROUP AsociaciónDeProductores = GrupoOviedo3, LSI

Otros tres constructores denotan exclusión (EXCEPT), diferencia (OR) e intersección (AND). Por ejemplo:

- Owner=Arturo EXCEPT manufacturer=Microsoft – todos los agentes enviados por Arturo pero no escritos por Microsoft.
- Owner=Arturo AND context=ServidorDeLABTOO – todo agente propiedad de Arturo y que llega de ServidorDeLABTOO.

La pequeña especificación del lenguaje más los privilegios soportados permiten la definición de la base de datos de políticas de administrador del contexto y de la base de datos de preferencias.

20.1.2 Los privilegios

Los privilegios definen las capacidades del código en ejecución al establecer restricciones de acceso y límites en el consumo de recursos. Un privilegio es un recurso junto con su permiso asociado. Algunos recursos contemplados son:

- File – ficheros en el sistema local.
- Net – acceso a la red.
- Awt – sistema de ventanas local.
- System – cualquier tipo de recurso del sistema.

Los recursos tienen una estructura jerárquica. Por lo que, los permisos se pueden dar a un conjunto de recursos específicos o inclusive a un tipo de recurso completo. Por ejemplo el acceso universal al sistema de fichero o bien a un fichero particular que se detalla a continuación.

- File – todos los ficheros.
- File /tmp/ejemplo.txt – sólo al archivo /tmp/ejemplo.txt.

El acceso a la red se considera un recurso más elaborado por lo que se pueden distinguir entre protocolos, servidores y puertos, por ejemplo:

- Net – cualquier tipo de operación de red.
- Net TCP – cualquier tipo de conexión TCP.
- Net TCP host port – conexiones TCP al host especifico pero sólo a través de puerto señalado.

Haciendo uso del lenguaje y combinando recursos con permisos, se pueden definir privilegios de la siguiente manera:

- Net TCP ServidorDeLABTOO 930-033 NOT connect – el aglet no puede conectarse al contexto ServidorDeLABTOO usando TCP en los puertos del 930 al 933.
- System MAX_MEMORY 12 – el aglet no puede almacenar mas de 12 Mbytes de memoria.
- AWT Top_level_windows 1 – el aglet puede crear una ventana.

20.1.3 La base de datos de políticas del administrador del contexto

El administrador del contexto define la política de seguridad para los contextos de aglets bajo su control. Esta política define las acciones que un agente puede hacer. En la base de datos, el administrador combina las autoridades que pueden denotar grupos de aglets y sus privilegios y la convierte en reglas. La forma sintáctica de una regla es:

<etiqueta> : <autoridad> -> <privilegios>

Cuando un aglet concuerda con muchas autoridades, se combinan las políticas para esas autoridades. En otras palabras, una regla negativa rechaza la petición. El contenido de una base de datos podría ser con el siguiente:

TRUSTED:
Manufacturer=Arturo OR master=AdministradorLABTOO ->
File /tmp read, write
Net TCP DominioLABTOO accept
Top_level_windows 3

GUEST:
Manufacturer=Oviedo3 ->
Net message EUITIO receive
Top_level_windows 1
System MAX_MEMORY 12

REJECT:
Manufacturer=Malevolo, Malicioso ->
Context NOT enter

20.1.4 Preferencias del propietario de los aglet

El propietario del agente tiene la oportunidad de establecer un conjunto de preferencias de seguridad que serán llevadas a cabo por los contextos que el agente pueda visitar. Las preferencias combinan grupos de contextos y privilegios en reglas. La forma sintáctica de la regla es:

<etiqueta> : <definición_de_grupos_de_contextos> -> <privilegios>

La siguiente lista define el conjunto de métodos que un propietario puede restringir sobre sus aglets.

- Clone/deactive/retract/dispose
- Send message
- Set Itineraty/property/text
- Subscribe/unsubscribe (all)messages

Las siguientes son ejemplos de preferencias de seguridad:

- Context= EUITIO EXCEPT master=AdministradorLABTOO -> ITINERARY set – el itinerario del aglet puede ser cambiado en el contexto EUITIO pero solo si no es administrado por AdministradorLABTOO.
- -> aglet=MyAglet OR owner=Arturo AGLET dispose – en cualquier contexto el aglet podría ser eliminado sólo por el aglet MyAglet o por algún otro aglet propiedad de Arturo.

20.1.5 Aplicación de SAHARA al modelo de seguridad de los Aglets

El modelo de seguridad de los aglets contiene varias deficiencias que pueden ser mejoradas si incorpora en dicho modelo parte de la filosofía de SAHARA, quizá si los aglets desean seguir el modelo de Günter Karjoth sólo podrán incorporar ideas claves de la arquitectura de SAHARA que les permita mejorar su eficiencia a la hora de implementarlo.

Quizá el primer inconveniente del modelo es la complejidad que involucra, lo que augura que sea poco factible una implementación total y si se llega a implementar sería muy costosa. A continuación serán mostrados algunos inconvenientes del modelo de seguridad de los aglets y cómo incorporando conceptos de la arquitectura de seguridad de SAHARA pueden mejorarse.

20.1.5.1 Gran cantidad de Autoridades

El modelo de los aglets involucra una gran cantidad de autoridades muchas de ellas difícil de distinguir. Peor aún algunas de sus autoridades se traslapan o son innecesarias. Por ejemplo el otorgar un permiso a la persona que diseño un programa (llamado productor del aglet) resulta innecesario desde que su propietario lo puede modificar antes de enviarlo.

Otra dificultad evidente es a la hora de la autenticación, ya que se supone que los permisos serán otorgados basándose en las autoridades de las que formen parte los aglets. El contexto y el propietario de un agente puede autentificarse mediante firmas digitales, pero ¿cómo saber que un agente pertenece al productor de un contexto?.

Si se dan permisos a ese productor del contexto, ¿cómo se valida que sea verdad que los agentes que clamen pertenecer a él en verdad pertenezcan?. El tener 8 autoridades tiene como finalidad brindar más granularidad a la hora de otorgar permisos pero resulta casi imposible su validación.

El modelo de los aglets puede adoptar la sencillez del modelo de SAHARA reconociendo sólo dos autoridades y un administrador que se encargue de establecer las políticas de seguridad. En la gran mayoría de las aplicaciones de agentes móviles es suficiente otorgar permisos basándose en la autoridad a la que pertenece el agente y en sistema del que proviene.

Es muy poco probable que se requiera una aplicación en donde se necesite otorgar permisos diferentes a sólo un agente de una autoridad y al resto permisos diferentes, por ello el número de autoridades que reconoce SAHARA resulta ser más apropiado y fácil de implementar.

20.1.5.2 Perdida de desempeño al utilizar un interprete para el lenguaje de seguridad

El utilizar tantas autoridades produce que cuando el agente quiera usar un recurso se ejecute el interprete del lenguaje de seguridad que tendrá que analizar el código de la base de datos de políticas y de las preferencias de usuario para poder definir si el aglet está o no autorizado para efectuar la acción que desea.

El modelo de los aglets puede adoptar la filosofía de SAHARA de crear dominios de protección en donde se enlazan los agentes con sus autoridades y estas a su vez con los permisos que les han sido otorgados en los ficheros de políticas de seguridad. Los dominios de protección se crean en demanda y sólo se crean la primera vez que llega un agente de una autoridad que no contaba con un dominio de protección establecido.

La sencillez en la definición de permisos de SAHARA y en la gestión de los mismos evita la perdida de la gran cantidad de tiempo que invertirá el interprete de los aglets en verificar todos los permisos que un agente puede tener al tener varias ocurrencias (debido a que puede formar parte de varias autoridades) en la base de datos de políticas.

20.1.5.3 Posibilidad del ataque de agotamiento de recursos

El propietario del agente puede definir las cantidades de consumo máximo de memoria RAM y espacio en disco que puede consumir cada uno de sus agentes. Sin embargo para proteger a un servidor de un ataque de agotamiento de recursos esta política no sirve de nada. Es decir, de nada sirve que un usuario ponga un límite en los recursos que pueden consumir sus agentes ya que el sistema receptor no lleva a cabo una validación de la suma total que dicha autoridad esta consumiendo.

En el modelo de los aglets una vez que una autoridad ha sido aceptada en un sistema no se le impone ningún límite en el consumo de recursos que sus agentes pueden tener, por lo que, en caso de que sea una autoridad que quiera causar un daño al sistema puede provocar el agotamiento de recursos simplemente enviando un gran número de agentes que agoten el límite de recursos que le ha sido establecido a cada uno de ellos.

Es necesario que en el sistema además de definir las preferencias del usuario, es decir, las cantidades máximas que él desea que sus agentes puedan consumir, exista una contraparte que se encargue de fijar un límite de recursos por autoridad y que luego se verifique que esos límites no son sobrepasados en el sistema. SAHARA tiene este control al imponer un límite en el consumo de asignaciones por autoridad remota y el modelo de los aglets podría adoptarlo fácilmente.

20.1.5.4 Política de Seguridad Estática

En el modelo de los aglets una vez definida la política de seguridad por el administrador del contexto y las preferencias por los propietarios de los aglets no puede efectuarse ningún cambio en tiempo de ejecución.

Este modelo puede adoptar la filosofía de SAHARA que ofrece un mecanismo versátil para modificar la política de seguridad en tiempo de ejecución ya sea para que los agentes puedan ofrecer ficheros decidiendo con quien quieren compartirlos[109] o para que el sistema restrinja permisos cuando note una escasez de ellos y pueda mantenerse estable.

Una política de seguridad Dinámica como la de SAHARA da la versatilidad de modificarla cuando se requiera y fijar el nivel de permisos de tal forma que resulte óptimo para mantener estable el sistema.

20.2 LA APROXIMACIÓN DE FRITZ HOHL PARA RESOLVER EL PROBLEMA DEL SERVIDOR MALICIOSO

Esta aproximación pretende evitar todos los ataques que un agente móvil podría sufrir de un servidor malicioso entre los que se encuentran:

- Espionaje del código.
- Espionaje de los datos.
- Espionaje del flujo de control
- Manipulación de los elementos anteriores.
- Falsificación.
- ♦ Ejecución incorrecta del código.

Según afirma Fritz [HOHL97] la mayor parte de estos problemas de seguridad pueden ser resueltos si el servidor no es capaz de determinar la relación entre las líneas de código y su semántica y la relación entre los bits de memoria y la semántica de los elementos de datos. Evidentemente el servidor malicioso podría modificar de igual forma el código y los datos pero no con un efecto computarizado correcto.

La idea básica de su aproximación consiste simplemente en no darle tiempo suficiente al servidor malicioso para hacerlo. Este puede ser logrado mediante una combinación de las siguientes ideas: revoltura de código y tiempo de vida limitado de código y datos.

20.2.1 Revoltura de código

Antes de la primera migración el servidor de origen crea una nueva forma del código del agente que sea la misma y haga exactamente lo mismo pero que sea vea y este distribuida de forma diferente. Al mismo tiempo, el servidor de origen toma los datos del agente, revuelve los elementos de datos originales y distribuye dicha revoltura en una lista de nuevos elementos de datos.

[109] Mediante el método share

Antes de iniciar la conversión se incluyen fechas de caducidad que se firman digitalmente para evitar manipulación. Finalmente, el servidor de origen firma el código del agente con otra fecha de caducidad y lo envía.

Para revolver el código se sigue una forma estructurada que consiste en ir en contra de las indicaciones para construir un programa legible, es decir:

- Se usan nombres de variables cortos, que no digan nada y que utilicen caracteres aleatorios.
- El agente no se crea modularmente por lo que todo el código se agrega en un todo en vez de crear subrutinas.
- Se escoge una representación de datos que haga al programa difícil.
- Se crean nuevas variables mezclando los bytes de las variables anteriores para que acceder a ellas sea más complicado.

20.2.2 Tiempo de vida limitado de código y datos

Al llegar a un servidor malicioso, éste analiza el código y la estructura del agente. Durante ese intervalo transcurrirán varios minutos. El problema para el servidor malicioso es que nunca antes había visto la forma de código que el agente tiene. Además las fechas de caducidad de los datos y del código caducan tan pronto y hayan pasado unos cuantos segundos.

Durante ese tiempo el servidor sólo puede ejecutar el agente y no le daría tiempo para analizar el código o sus datos para hacer alguna alteración a favor de sus malévolos intereses. Si lo hace, las fechas de caducidad habrán terminado y cuando el agente sea enviado a otro servidor será rechazado inmediatamente por el nuevo receptor.

El tiempo permitido va introducido en la fecha de caducidad que es puesta en la firma digital, la contraparte de la firma no será válida después de este periodo de tiempo. Otra posibilidad sería fijar un tiempo constante después del cual las parte relevantes del agente fueran inválidas.

20.2.3 Análisis de la aproximación y cómo puede ayudar SAHARA al modelo de seguridad de MOLE

La aproximación creada por Fritz Hohl presenta varias dificultades que conllevan sobre todo a una difícil implementación, presenta las siguientes deficiencias:

- Inicialmente la creación de un reconstructor de código que se encargue de revolver el código en la forma necesaria para que sea ilegible y fraccionando los datos en estructuras complejas no viene a ser fácil.
- El algoritmo para distribuir las llaves públicas que se utilizará la autentificar al agente y definir el periodo de validez tampoco es fácil de implementar. Se requiere que exista un mecanismo rápido y eficiente de distribución de claves públicas para

que los sistemas pudieran tener las nuevas claves de forma inmediata y los agentes que llegarán en un futuro a esos sistemas pudieran ser validados inmediatamente.
- Muchas aplicaciones de agentes móviles no podrían ser implementadas si los agentes móviles sólo pueden estar por unos segundos.
- Finalmente la aproximación de Fritz ha sido criticada ya que muchas personas consideran que un agente móvil al estar a merced de un servidor malicioso éste puede encargarse de modificar la hora de su máquina o hacer más largo el periodo de validez con el fin de poder descifrar el código revuelto del agente.

Actualmente el equipo de investigación de seguridad en sistemas de agentes móviles de la Universidad de Stuttgart no ha implementado un mecanismo de seguridad eficiente. Actualmente dentro de su modelo cuentan con dos tipos de agentes. Los que son móviles representan las aplicaciones de los usuarios mientras que los que son estáticos se encargan de contribuir al funcionamiento y seguridad del sistema, éstos últimos son los que se encargan de autorizar a los agentes móviles provenientes de otros sitios de que puedan o no realizar una tarea o de proveerles de algún recurso en particular.

Ellos se enfocaron en el problema más difícil de resolver sin haber encontrado una solución satisfactoria para los problemas más sencillos, por ello su sistema MOLE como fue escrito en Java puede beneficiarse ampliamente de la arquitectura de SAHARA.

Pueden utilizar el concepto de regiones y grupos de regiones para tratar de crear redes seguras en donde los mecanismos de seguridad sean innecesarios, de igual forma pueden adoptar el protocolo de transferencia de SAHARA que utiliza SSL que permite la autenticación mutua de servidores.

De igual forma para proteger los datos del agente pueden usar los códigos de autenticación de resultados parciales descritos en la arquitectura de SAHARA. Pueden beneficiarse igualmente que los aglets de todos los puntos descritos en el subtema 20.1.5 y de los que se describen en el tema siguiente.

20.3 APLICACIÓN GENERAL DE SAHARA A OTROS SISTEMAS DE AGENTES MÓVILES

Cómo SAHARA ha sido diseñado para aprovechar las máximas ventajas de Java, sus beneficios pueden ser mejor aprovechados por los sistemas de agentes móviles basados en Java como Corcordia, Odissey, Voyager, Mole, etc.

Como se mencionó al principio de este capítulo el trabajo referente a seguridad implementado por otros sistemas de agentes móviles es demasiado elemental por lo que muchos sistemas podrían incorporar varios de los puntos que se indicaron en el apartado 20.1.5 llamado aplicación de SAHARA al modelo de seguridad de los aglets.

Adicionalmente a lo descrito en el punto 20.1.5 la mayor parte de los sistemas actuales pueden incorporar algunas de las ideas innovadoras que han sido introducidas en SAHARA.

20.3.1 Agentes Modulares y Compresión

En la arquitectura de SAHARA se introdujo el concepto de modular las partes que constituían un agente, en una parte se guarda su estado de código que sería firmado sólo una vez al salir del servidor original. En otra parte se guardaba su estado de datos serializado que se firmaría cada vez que el agente migrará de un servidor a otro.

Finalmente en otro directorio dentro del fichero Jar se guardaban las firmas digitales y el fichero *manifest* que contiene toda la información referente a las autoridades que habían firmado las diferentes partes del agente previamente.

Estos dos conceptos pueden ayudar a mejorar el desempeño de los sistemas actuales, por una parte se agiliza el envío al comprimir un agente y además se facilita la carga, validación y gestión de los estados del agente al estar estos guardados en carpetas diferentes.

20.3.2 Restricción del Acceso al Sistema

Los sistemas actuales permiten el acceso libre al sistemas a cualquier usuario que pueda hacer uso del ordenador donde se encuentre el servidor, esto conlleva a la existencia de un cierto peligro de que algún usuario por maldad o error modifique la configuración del sistema afectando directamente la seguridad del mismo.

La mayor parte de los sistemas pueden adoptar de SAHARA no sólo el concepto de validar el acceso al sistema mediante una y una clave secreta sino que pueden hacer, como propone SAHARA, que las cuentas de acceso se validen contra el *keystore* para evitar el mantenimiento de dos repositorios de llaves protegidos. De esta forma los usuarios/autoridades que pueden enviar agentes son los que pueden ingresar al sistema.

20.3.3 Autenticación Múltiple

La separación de estados en distintas carpetas permite la autenticación múltiple de agentes, es decir, un sistema podrá otorgar permisos a los agentes cuyo estado de código haya sido firmado por Arturo y por Darío. O bien otorgar permisos al agente cuyo estado de datos haya sido firmado por el ServidorLABTOO y por el grupo de investigación de Oviedo3.

Las firmas digitales múltiples habilitan la posibilidad de una autenticación múltiple lo cual puede ser usada por los sistemas actuales para otorgar permisos basándose en más de una autoridad.

20.3.4 Firmas de los Servidores para Adjudicar Responsabilidad

Hoy en día ningún sistema de agentes móviles utiliza un mecanismo para proteger el estado de datos de sus agentes debido a la dificultad de implementación que conlleva dicho mecanismo. Los sistemas que desean agregar el mecanismo propuesto por SAHARA de PRAC's podrán contar con la última versión del agente antes de que este haya sido alterado.

Si no desean incluir ningún mecanismo en sus implementaciones pueden incorporar en sus sistemas que el estado de datos de los agentes sean firmados por la autoridad del servidor al que pertenecen para que dicha autoridad ofrezca una garantía de que el no altero los datos del agente, y en caso de que lo hubiera hecho poder adjudicarle una responsabilidad legal sobre dicha alteración.

20.3.5 Seguridad en la transmisión mediante SSL

Muchos sistemas podrán adoptar el protocolo de transmisión que utiliza SAHARA basado en SSL ya que permite la autenticación mutua de servidores lo que permite abrir una conexión con una seguridad plena de que al servidor a donde se están enviando los agentes es seguro.

El protocolo utiliza claves para la autenticación en el establecimiento de la conexión y posteriormente inicia una conexión por donde transmite la información encriptada. Este protocolo ya ha sido incorporado en el sistema de Ara y de Concordia.

20.3.6 Uso del concepto de región para crear redes confiables

Desgraciadamente la seguridad tiene un precio muy alto en desempeño, cada vez que se firma un agente, el tiempo que se pierde en enviarlo firmado de 10 a 20 veces mas lento que si se envía sin firmar lo que ralentiza el desempeño de la aplicación y del sistema.

SAHARA introduce el concepto de región y de grupos de regiones pero no sólo para agrupar servidores sino para que dicha agrupación sea empleada con fines de seguridad. Es decir, todos los agentes que sean enviados dentro de una misma región o dentro de un grupo de regiones no serán firmados para ahorrar la gran cantidad de tiempo que se pierde en dicho proceso, los servidores sólo serán autenticados antes de iniciar la transmisión.

Adicionalmente dicho concepto puede ser implementado de una forma versátil de tal forma que permita que los agentes al migrar dentro de una misma región sean enviados utilizando el protocolo SSL en su transferencia o inclusive sin él en las regiones más seguras.

Capítulo XXI
EVALUACIÓN Y AMBITOS DE APLICACIÓN DE MILENIO

Concluido el primero prototipo de nuestro sistema es importante compararlo con algunos de los sistemas actuales para observar las características que poseen y poder sugerirlo para un tipo de aplicaciones específicas de donde se tome el mayor provecho de las ventajas que posee.

Es importante mencionar que MILENIO no ha sido diseñado con el fin de superar todas las bondades que ofrecen los sistemas de agentes móviles comerciales actuales, sino para demostrar el buen funcionamiento de la arquitectura de seguridad SAHARA, MILENIO es por lo tanto un sistema vanguardista y mejor en aspectos de seguridad, pero inferior en otros aspectos[110].

Una vez que se ha definido apropiado para una gama de aplicaciones para las que resulta óptima debido a su eficiente sistema de seguridad, es importante señalar la variedad de aplicaciones donde se le puede sacar mayor provecho al sistema.

Aunque algunas de las aplicaciones mencionadas son simples aplicaciones de agentes no podrían llevarse a cabo de una forma confiable y eficiente si no tuvieran el soporte como el que ofrece MILENIO, es decir, de un sistema de agentes móviles seguro.

21.1 EVALUACIÓN Y COMPARACIÓN DE MILENIO CON OTROS SISTEMAS

Debido a la heterogeneidad de los sistemas de agentes móviles resulta difícil definir un baremo apropiado para todos los sistemas porque cada día nacen nuevos sistemas de propósito particular para los que muchos de los parámetros definidos en el barémo no apliquen.

[110] Por ejemplo en los medios para la comunicación entre agentes ya que MILENIO no proporciona en esta versión ningún soporte.

No obstante para sistemas de agentes móviles de propósito general se pueden definir algunas características que son muy importantes para definir la calidad y eficiencia de un sistema. A diferencia de otro tipo de programas la velocidad no es un factor fundamental ya que se considera que las aplicaciones de agentes móviles suelen ser autónomas. Entre las características a resaltar se pueden mencionar en orden de importancia:

- Portabilidad.
- Seguridad.
- Lenguajes de programación de agentes que soporta.
- Comunicación.
- Interfaz gráfica con el usuario.
- Velocidad.

Dentro del grupo de los sistemas portables destacan los desarrollados en Java ya que es posible ejecutarlo bajo cualquier plataforma que soporta la JVM, entre ellos se puede citar a los Aglets, Concordia, MILENIO y Voyager que destaca porque brinda además las características de un ORB CORBA. Estos sistemas son por supuesto, más portables que aquellos que son ejecutados en lenguajes dependientes de una sola plataforma como D'Agents y Ara.

Por su parte D'Agents y Ara son superiores a los basados en Java porque permiten que los agentes sean programados en más de un lenguaje, lo que da una gran versatilidad a la hora que desarrollar aplicaciones.

No obstante como el objetivo principal de este trabajo ha sido desarrollar una arquitectura de seguridad integral para sistemas de agentes móviles y la seguridad es un factor fundamental dentro de las aplicaciones móviles, el análisis comparativo de los sistemas de agentes móviles será en torno a sus características de seguridad.

21.1.1 Análisis comparativo de la seguridad de MILENIO contra otros sistemas de agentes móviles

La seguridad de un sistema de agentes móviles está en función de las garantías que pueda ofrecer a las autoridades/propietarios tanto dichos sistemas como de los agentes que intervengan en una aplicación de que nadie va a dañar, robar o alterar los recursos con que cuentan, ya sea por ataques mutuos o por terceras partes.

Debido a ello, las medidas de seguridad que ofrezca un sistema se dividen en tres grandes categorías:

- Medios para proteger al servidor de agentes y sus recursos.
- Medios para proteger la transmisión de los agentes móviles.
- Medios para la protección del código y datos de los agentes.

Aunado a ellas es importante considerar también la versatilidad que se tiene a la hora de aplicar las medidas de seguridad. De estas tres categorías se derivan once medidas de seguridad que un sistema de agentes móviles debiera tener para lograr una seguridad integral. Éstas se presentan a continuación.

21.1.1.1 Las características necesarias que todo sistema debiera tener para ser seguro

1. *Autenticación de agentes:* puede llevarse a cabo de dos formas, global separando a los agentes en locales y remotos o particular identificando a cada autoridad propietaria de agentes para asignarle recursos.
2. *Autenticación de servidores:* debe realizarse antes de abrir la conexión para realizar el envío de los agentes, lo que permite tener la garantía de que se están enviando los agentes a un sitio seguro.
3. *Asignación de privilegios/recursos:* al igual que el primer punto puede ser global o individual, lo ideal resulta ser individual. Además es deseable también que se puedan precisar los permisos, por ejemplo indicar a que archivos del sistema de ficheros se puede acceder y con que privilegios.
4. *Verificación del consumo de recursos por agente:* se debe supervisar que el agente no consuma más de lo que se le ha autorizado.
5. *Verificación del consumo de recursos por autoridad:* además de contabilizar lo que consume cada agente es necesario que se contabilice el consumo total de todos los agentes que pertenecen a una autoridad para que no se les permita sobrepasar el límite que se le ha impuesto a dicha autoridad.
6. *Verificación automática de los recursos disponibles del sistema:* cada sistema debe saber en todo momento los recursos con los que cuenta para que pueda tomar las medidas pertinentes y evitar venirse abajo.
7. *Modificación de la política de seguridad en tiempo de ejecución:* es necesario que cuando el servidor de agentes carezca de recursos pueda modificar la política de seguridad que se encuentre en vigor para evitar un agotamiento total y el sistema pueda seguir estable.
8. *Protección de los recursos que genere el agente:* si un agente genera un fichero, un mensaje o una conexión de red esta deberá ser incluida en la política vigente para que sea protegida de acuerdo a los privilegios de su autoridad.
9. *Transferencias de agentes segura:* debe usarse un protocolo de transferencia segura que encripte la información y no pueda ser falsificada o alterada durante su envío.
10. *Restricción de acceso al sistema:* el sistema de agentes deberá validar el acceso de los usuarios mediante una cuenta y clave de acceso para evitar usuarios indeseados.
11. *Protección del código del agente:* el programa principal y las clases requeridas para reanudar al agente en un servidor remoto deben ser protegidas.
12. *Protección de los datos del agente:* el estado de datos del agente deberá ser protegido contra posibles ataques de un servidor malicioso.

MILENIO como se presenta a continuación cuenta con la mayor parte de las características mencionadas ya que están incluidas en la arquitectura de seguridad de SAHARA.

21.1.1.2 Comparativa de la seguridad de MILENIO contra otros sistemas de Agentes Móviles

Una vez descritos los parámetros de evaluación compararemos las características de seguridad de MILENIO con las de los sistemas de agentes móviles estudiados en el capítulo VIII que son D'Agents, Ara y los Aglets. Incluiremos además en la comparativa al sistema de Concordia que es un sistema de agentes comercial basado en Java que cuando salió al mercado hace un par de años fue considerado el sistema de agentes móviles basado en Java más seguro.

En importante indicar que la tabla comparativa que a continuación se presenta considera las características presentes en las últimas versiones disponibles de estos sistemas, ya que algunos sistemas observando sus deficiencias de seguridad han anunciado futuras mejoras que serán establecidas (como el modelo de seguridad de los Aglets) pero que sin embargo aún no se encuentran presentes. Cada sistema tiene una S si proporciona dicha característica en caso contrario tendrá una N. Si la proporciona parcialmente tendrá una P.

SISTEMA DE AGENTES MOVILES	Autenticación de Agentes global	Autenticación de Agentes Individual	Autenticación de Servidores	Asignación de Recursos Individual	Verificación del Consumo por Agente	Verificación del Consumo por Autoridad	Verificación de los Recursos del sistema	Cambio de la Política de Seguridad	Protección de Recursos generados por el agente	Transferencia Segura	Validación de Acceso	Protección del Código	Protección de los Datos
D'Agents	S	S	N	P[111]	S	N	N	N	N	N	N[112]	S	N
Ara	N	N	N[113]	P[114]	S	N	P[115]	N	N	N[4]	N	N	N
Aglets	S	N	N	N[116]	N	N	N	N	N	N	N	N	N
Concordia	P	S	S	S	N	N	N	N	N	S	N	S	N
Milenio	S	S	S	S	S[117]	S[8]	S[8]	S	S[118]	S	S	S	N

[111] Logra asignar listas de acceso de forma individual, pero no puede indicar a que elementos del recurso se puede acceder. Por ejemplo sólo puede permitir o no el acceso al sistema de ficheros pero no puede particularizar a que ficheros se permite el acceso.
[112] La restricción de acceso puede hacerse mediante permisos unix pero no a través del sistema
[113] La autenticación de servidores y la transmisión segura se logran al incluir SSL en el protocolo de transferencia el cual esta siendo implementado en Ara pero aún no se encuentra disponible.
[114] Al igual que en D'Agents solo se controla el acceso al recurso pero no es posible particularizar los elementos del recurso.
[115] Realiza verificaciones sólo sobre el consumo de memoria.
[116] Solo asigna los privilegios a los dos grandes grupos que autentifica, locales y remotos.
[117] Verifica los que permite la máquina virtual de Java actual, no puede contabilizar espacio en disco consumido por agentes ni la cantidad de memoria RAM. No obstante la verificación y el control de los recursos del sistema se lleva a cabo automáticamente.

Como se observa SAHARA abarca casi en su totalidad el amplio espectro de la seguridad. Además algunas de las características que proporciona lo hace con más versatilidad que como lo llevan a cabo algunos otros sistemas. Si se resumen los resultados en una tabla contabilizando las características de seguridad que poseen se tiene:

Sistema de Agentes Móviles	Número de Características de Seguridad
D'Agents	4.5
Ara	2
Aglets	1
Concordia	5.5
Milenio	12

Y de estos resultados se puede hacer una pequeña gráfica que muestre que sistema cuenta con una mayor capacidad en seguridad actualmente.

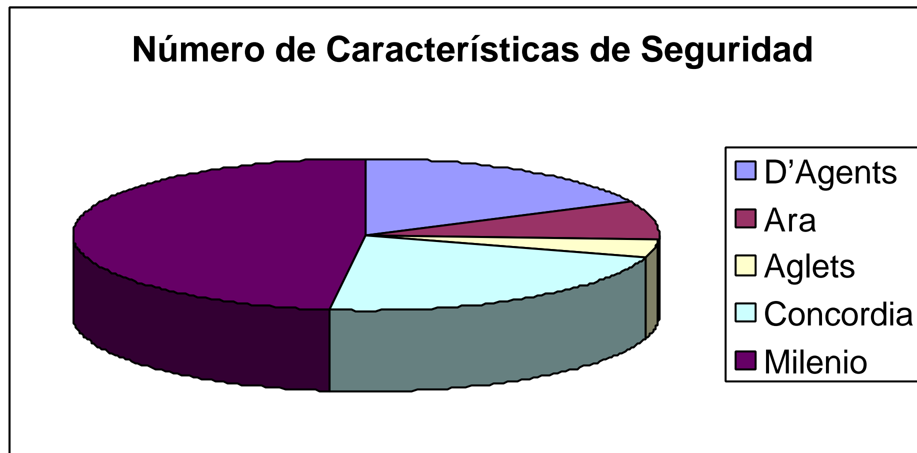


Figura 21.1.1.2.1 – Capacidad de seguridad de los Sistemas Analizados

Como se aprecia la relación la implementación de MILENIO tiene una cobertura de seguridad más amplia que el resto de los sistemas. No obstante la seguridad tiene un precio muy alto por lo que seguramente el mejor desempeño lo tendrá el sistema de los aglets quien prácticamente cuenta con el mecanismo más básico de seguridad.

Las firmas digitales que permiten la autenticación particular de agentes así como la protección de su código desgraciadamente tienen la gran desventaja de disminuir enormemente el tiempo de envío ya que se estima [LANG97] que puede llegar a ser 100 veces más lento que si no se firmara. Se presenta pues una evaluación del sistema más seguro y el más inseguro que se ejecutan sobre una misma plataforma para ver el precio que tiene la seguridad.

[118] Protege sólo los ficheros generados por los agentes que se incluyen en tiempo real a la política de seguridad mediante el método *share*.

21.1.2 El Precio de la Seguridad en el Desempeño del Sistema

Como es obvio suponer cada característica adicional de seguridad que tenga un sistema exige una comprobación más que lleva tiempo y afecta en el desempeño. No obstante, MILENIO ha sido diseñado mediante técnicas bien estudiadas que en todo momento tratan de favorecer el desempeño de la aplicación.

Otro gran problema representa el cómo evaluar el desempeño de un sistema de agentes móviles ya que no se puede contabilizar el tiempo que tarda el agente al ser transmitido a través de la red de área amplia ya que siempre habrá diferencias al ejecutar pruebas diferentes en tiempos diferentes aunque se tratase de la misma red. Obviamente las diferencias podrían ser enormes si se prueba la velocidad de ida y vuelta de los agentes a dos servidores que se encuentran en países distintos.

Debido a ello lo más importante a evaluar en dichos sistemas para tener una idea en su desempeño será el tiempo de creación, el tiempo de migración[119] así como el tiempo de reanudación de un agente que será el tiempo que tarda desde que se recibe el fichero hasta que se reanuda la ejecución de agente.

Para evaluar el desempeño de MILENIO realizamos una prueba comparativa con el sistema de agentes móviles de los aglets, es decir creamos un agente similar en ambos sistemas y medimos con total precisión los tiempos antes mencionados. Las pruebas las realicé en una vieja LAN de ordenadores Pentium MMX a 200 Mhz con 64 MB de RAM y en condiciones similares. Los resultados de los promedios obtenidos de las tareas probadas dados en milisegundos pueden apreciarse en la siguiente tabla.

Sistema de Agentes	*Creación Primer Agente*	*Creación Sig. Agentes*	*Migración Primer Agente*	*Migración Sig. Agentes*	*Migración Sig. Agentes Sin Firmar*
Aglets	140ms	18ms	5398ms No Firmado	4567ms No Firmado	4567ms
MILENIO	420ms	15ms	15383ms Firmado	11627ms Firmado	2300ms

Como se esperaba se aprecia una perdida de desempeño en el sistema, no obstante inferior a lo que se esperaba por la gran lentitud supuesta en la criptografía de clave pública descrita en [LANG98b].

La creación y migración de un agente la primera vez es mucho mayor a las siguientes debido a que el servidor tiene que leer las clases involucradas en dichos procesos desde ficheros, que para las veces sucesivas suelen estar ya en la memoria caché.

MILENIO es más lento la primera vez en el tiempo de creación porque tiene que abrir todos los ficheros de donde toma las asignaciones y objetos de seguridad que llevará el

[119] Este tiempo será el tiempo requerido por un agente para estar listo para ser enviado, involucra el tiempo de compresión, serialización y firmado (en caso de existir).

agente, por asignar al agente al dominio de protección que le corresponda, y por otras tareas que no realiza el sistema de los aglets. Las creaciones sucesivas de agentes son más rápidas en MILENIO aunque presentan un comportamiento muy similar en ambos sistemas.

Encontramos a MILENIO considerablemente más lento sólo en el tiempo de migración cuando los agentes son firmados. Esto es debido principalmente a la cantidad de tiempo que emplea el algoritmo matemático que firma digitalmente al agente, además MILENIO antes de serializar al agente lo comprime con lo que invierte tiempo adicional en dicha tarea pero logrará que la transferencia a través de la red sea más rápida.

El doble de tiempo que invierte MILENIO en esta tarea se debe pues a que él serializa, comprime y firma, mientras que los Aglets sólo serializan, no obstante, cuando los agentes no son firmados MILENIO resultó ser un poco más rápido.

Adicionalmente medimos el tiempo de reanudación de un agente que es el tiempo que tarda el servidor en ejecutar su instancia a partir de cuando recibe el fichero que contiene el agente. El promedio de MILENIO fue 2,73 segundos. El promedio de los aglets no lo pudimos medir con exactitud por no contar con su código fuente y carecer de un método que nos permitiera saber con exactitud el momento preciso en que el servidor recibe el fichero, realizando una estimación con un cronómetro el promedio fue de 2,1 segundos.

Aunque el proceso de reanudación de MILENIO fue un poco mas lento es mucho más seguro y eficiente, porque al llegar un agente es descomprimido, validada su firma digital y asignado a un dominio de protección basándose en su autoridad, los privilegios le son asignados del fichero de políticas de seguridad, mientras que los aglets sólo reanudan su ejecución.

Por último el tiempo de trasferencia por la red es más rápido en MILENIO debido a que los agentes son comprimidos. No obstante la diferencia sólo se puede apreciar cuando el tamaño del código de los agentes es considerable.

MILENIO sólo tiene una perdida de desempeño del doble en tan sólo una de las tareas del sistema de agentes, es decir, en la migración del agente, no obstante, las tareas que se realizan en este proceso son las que le brindan seguridad a los agentes móviles. En las tareas de creación y reanudación no hay una perdida notoria.

Como conclusión se puede afirmar que en los ordenadores más actuales la diferencia en transferencia de agentes de cinco segundos se ve reducida a dos, adicionalmente las aplicaciones de agentes móviles requieren mayoritariamente seguridad e independencia más que rapidez por lo que resulta muy conveniente usar un sistema como MILENIO y pagar esa ligera perdida de desempeño con el fin de poder realizar aplicaciones seguras.

21.2 ÁMBITOS DE APLICACIÓN DE MILENIO

MILENIO es uno de los sistemas de agentes móviles basados en Java más seguros en la actualidad y quizá el más seguro de los que soportan Java2. A continuación se describirán algunas de las aplicaciones en donde se puede sacar un mayor provecho del uso de este sistema de agentes móviles.

21.2.1 Aplicaciones donde se paga el acceso individual a un recurso

Estas aplicaciones son útiles para empresas que ofrecen una diversa gama de servicios o recursos y cuyo acceso a cada uno de ellos requiere un pago individual. Cada individuo se da de alta en la empresa solicitando y pagando los recursos o servicios que desea adquirir.

Como MILENIO proporciona una base de datos de políticas de seguridad en donde se establecen los permisos que tendrá cada usuario/autoridad que pueda acceder al sistema resulta ideal para llevar a cabo este tipo de aplicaciones.

Un ejemplo muy típico de este tipo de aplicaciones son las bibliotecas digitales en donde se paga por acceder a los libros de una o más materias. Los usuarios al contratar el servicio son dados de alta en el *keystore* incluyendo su certificado de clave pública para su validación y son agregados en la política de seguridad donde se permite el acceso a los libros de las materias contratadas. Esto se puede apreciar en la siguiente figura:

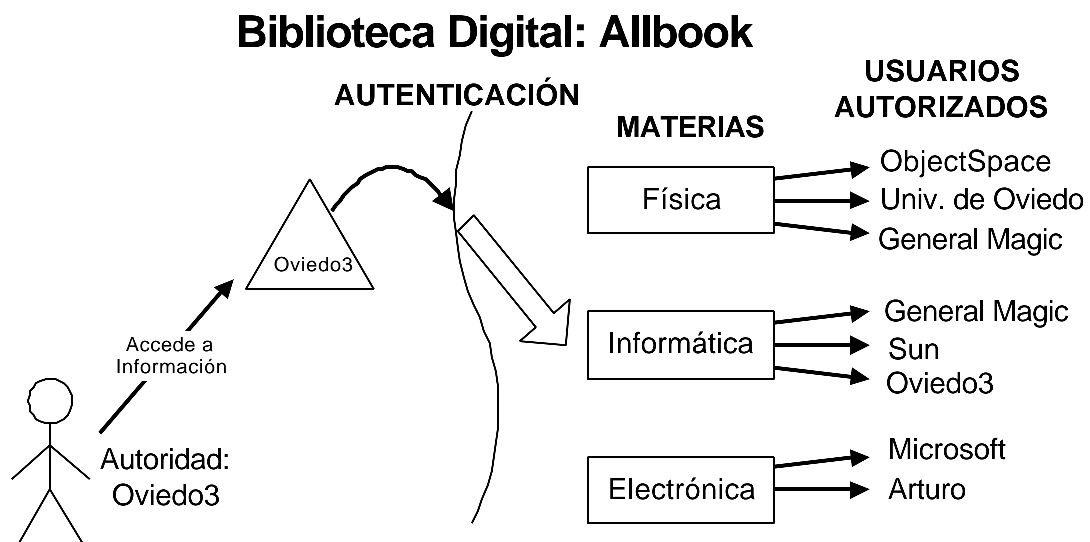


Figura 21.2.1.1 – Acceso Automático a Bibliotecas Digitales Mediante Agentes Móviles

En la figura la autoridad Oviedo3 había contratado previamente el acceso a los libros de informática por lo que envía a un agente por información a la biblioteca, a su llegada es autenticado con el certificado de clave pública y si es aceptado es dirigido sólo a los libros que éste puede acceder. Obviamente este modelo puede usarse a cualquier tipo de aplicación que siga el mismo patrón, es decir puede utilizarse en:

- *Ciber Vídeo clubs:* en donde se puede acceder a películas de un género contratado como sería terror, suspenso, acción, etc.
- *Licencias de software:* donde un usuario puede acceder a las versiones más actuales de software de una empresa de acuerdo a las licencias que haya contratado.
- *Colecciones:* que pueden ser de música, de fotografías u otros tipos. En ellos se contrata la clase de música o fotografía a las que se desea acceder.

21.2.2 Sistemas de Comercio electrónico

Uno de los grandes inconvenientes del comercio electrónico por medio de agentes móviles era la falta de seguridad a la hora de las negociaciones, es decir, cuando una agente móvil quería comprar un artículo en un centro comercial electrónico, ¿cómo tenía la certeza de que el agente que se lo ofrecía en realidad pertenecía a la empresa que decía representar y además le ofrecía el precio real?

El sistema de autenticación de MILENIO favorece el comercio electrónico ya que por medio de él se puede restringir el acceso al centro comercial.

21.2.2.1 Verificación y restricción de acceso al mercado electrónico

El servidor en donde se van a llevar a cabo todas las negociaciones puede incluir una lista de clientes y proveedores de productos en su *Keystore* de tal forma que puedan ser autentificados antes de ingresar al sistema. Cada cliente y proveedor que quiera negociar en el centro comercial electrónico deberá enviar su certificado de clave pública para ser dado de alta en el *Keystore*. De esta forma se evita que participen en las negociaciones entidades anónimas que quieran engañar a los participantes al ofrecer ofertas inexistentes.

21.2.2.2 Consumación de operaciones

Los agentes móviles pueden ir dotados de inteligencia o de una política de compra de bienes o servicios. Debido a ello, podrían adjuntar información bancaria para la consumación de operaciones como los datos de una tarjeta de crédito, de esta manera se podría efectuar el cobro directamente.

Como en MILENIO antes del envío de un agente se autentifica el servidor remoto cuando menos el dueño del agente sabe en manos de que autoridad esta dejando al agente, y en caso de ser reputación dudosa o desconocida puede no efectuar el envío del mismo. No obstante en la versión 2 de MILENIO se incorporará un mecanismo de protección de datos del agente con lo que se elimina el riesgo de que un servidor malicioso (en este caso un centro comercial electrónico) quiera hacer mal uso de la información del agente.

En la transmisión del agente y sus datos bancarios no se correría ningún peligro ya que el protocolo de envío de MILENIO se protege con SSL.

21.2.3 Operaciones Bancarias

Aunque muchas de estas operaciones se pueden realizar actualmente con los conocidos sistemas de “El banco en su casa” que ofrecen muchos bancos, estos sistemas presentan los siguientes inconvenientes:

- Autenticación manual cada vez que se desea acceder al sistema mediante un código de usuario[120] y un número de identificación personal (PIN). Este último número que coincide con el código de acceso a los cajeros inicialmente no es actualizado si el usuario lo cambia en un cajero automático, dificultando más el medio de acceso.
- Todas las operaciones que se realizan por un usuario son manuales y no tiene ningún medio de automatización para las tareas repetitivas.

Un sistema de agentes móviles seguro ofrece además de una autenticación automática y transparente, la posibilidad de automatizar tareas mediante la programación de varios agentes que lleven a cabo tareas repetitivas para un usuario o entidad determinada como las operaciones que se citan a continuación.

21.2.3.1 Operaciones seguras, programables y autónomas

Con el uso de un sistema de agentes móviles como MILENIO cada cliente del banco que solicitara el servicio podría ser dado de alta en el *Keystore* (incluyendo su certificado de clave pública) y en la base de datos de políticas de seguridad para definir su acceso (el cual sería sólo a todas las cuentas que tenga en el banco).

Las ventajas de usar el sistema de agentes son que el agente no requeriría autenticación manual para acceder a sus recursos ya que el sistema se encargaría de validarlo en forma automática y transparente con los que se tendría una completa seguridad en sus transacciones.

Además podría hacer operaciones autónomas y programadas en el tiempo. Por ejemplo, un inversionista de bolsa de valores, podría salir de vacaciones y dejar programados varios agentes para que a una hora determinada, de un día específico, invirtiera o retirara acciones de un fondo determinado de acuerdo a las fluctuaciones que él prevé para el futuro.

Otro ejemplo muy simple de las transacciones bancarias automatizadas que se realizarían con un agente programado sería el pago de nómina de una empresa a sus trabajadores en donde hacerlo a través de Internet manualmente resulta una tarea laboriosa.

[120] Normalmente de 12 dígitos de longitud e impreso en la tarjeta.

21.2.3.2 Transacciones inmediatas

El sistema de agentes puede a su vez ser usado por la entidad bancaria para actualizar la red de cajeros automáticos[121] y los usuarios puedan hacer disposiciones inmediatas en efectivo de las transacciones que se realizaran momentos antes.

De esta forma se haría posible lo que hoy en día resulta imposible, incrementar el saldo de una cuenta de débito desde América a través de Internet y poder disponer de ese dinero inclusive un día antes de la transacción en Europa si la diferencia horaria coincide.

21.2.4 Distribución selectiva y certificada de información

MILENIO proporciona la infraestructura para la distribución selectiva y certificada de información. Selectiva se refiere a que una aplicación de agentes tendrá la suficiente inteligencia para entregar la información sólo a las personas indicadas y certificada se refiere a que es posible obtener las garantías de que la información proviene de una entidad fiable mediante las firmas digitales, de que no ha sido alterada durante el envío mediante las firmas y SSL, y de que la información ha sido entregada cuando el agente regrese de su itinerario. Esto puede ser útil para aplicaciones como:

21.2.4.1 Distribución de programas y actualización de componentes

Una empresa podría repartir sus productos creando un agente que proporcionara un programa a todos aquellos clientes que le han pagado una licencia del mismo o bien proporcionarle los componentes de actualización a los clientes que tengan derecho a ella.

En ambos casos se requiere no sólo seguridad en la transmisión sino que el cliente receptor del software tenga una garantía que los programas recibidos le han sido enviados por la compañía a la que compro los productos, dicha garantía puede hacerla efectiva mediante la validación de su firma digital. Por la otra parte, una vez entregados los productos el agente en su estado de datos modificará las variables que indiquen que software ha sido entregado para prevenirse de futuras reclamaciones por parte de los clientes.

21.2.4.2 Entrega de correo electrónico certificado y acuse de recibo

[121] Siempre y cuando el hardware de los cajeros que suele ser muy heterogéneo soportara la JVM para que pudiera ser instaldo MILENIO.

Los lectores de correo electrónico ya incorporan firmas digitales para la autenticación de los emisores con lo que se puede tener la certeza de que un correo en realidad proviene de quien dice ser.

Sin embargo, al usar un sistema de agentes seguro, cualquier aplicación de agentes además de éste benefició puede hacer entrega selectiva de ese correo e inclusive el agente modificará su estado de datos para que a su regreso pueda informar a su emisor a quien fue capaz de entregarle ese correo.

El típico ejemplo de esta aplicación es el de un gerente de una gran corporación que quiere enviar un correo urgente para citar a una reunión a todos los directores de las sucursales de las poblaciones vecinas. Dentro de su agenda tiene a los directores, jefes y empleados de confianza por lo que el agente entregará el correo selectivamente, sólo aquellos cuyo cargo sea director. Además al recibirlo los directores podrán certificar que en realidad lo envío el gerente de la corporación, y además el agente le notificará al gerente a quien entregó el correo para que ningún director pueda excusarse de no haber sido informado.

21.2.5 Administración de Servidores y Balanceo de Tráfico en Grandes Redes

La administración de muchos servidores conectados en red que pertenezcan a una misma corporación pueden administrarse remotamente por un administrador que envíe agentes en su representación a hacer las tareas necesarias ya sea en secuencia o en paralelo.

Para que un agente móvil pueda modificar las configuraciones de los sistemas en necesario que sea autentificado eficazmente, de la misma forma que fue descrito en el punto 21.2.1, que su autoridad goce de privilegios para efectuar los cambios necesarios e inclusive compartir ficheros. Todo esto puede ser implementado fácilmente con el modelo de seguridad de MILENIO.

Si el administrador desea que todos los sistemas tengan la misma configuración bastara con que envíe un agente que recorra todos los servidores en secuencia y modifique dichas configuraciones de forma inmediata. Si existe más de una configuración pueden enviar tantos agentes como configuraciones existan a la vez, definiéndoles un itinerario específico a cada uno de ellos.

El tráfico en las redes también puede ser monitorizado por el sistema de administración basado en agentes para que posteriormente se modifiquen las configuraciones como se indicó anteriormente y se permite la mayor fluidez dentro de la red.

21.2.6 Aplicaciones Diversas que exijan seguridad

Dentro de este tipo de aplicaciones entran todas las descritas en el capítulo VII pero que adicionalmente requieran un grado de seguridad considerable en el acceso o uso de

recursos que haga indispensable el uso de un sistema de agentes móviles seguro como MILENIO. Por ejemplo:

21.2.6.1 Aplicaciones concurrentes

Aplicaciones que requieran de varios agentes trabajando concurrentemente que procesen parte de la información a la vez y los ficheros de salida los compartan mediante el método *share* para que otro agente de la misma autoridad pueda ir recopilando los resultados. La información a procesar se supone que es confidencial por lo que es indispensable que los agentes compartan la información sólo entre los agentes de la autoridad para que ningún otro agente pueda acceder ella.

21.2.6.2 Uso moderado de recursos de una supercomputadora

Procesos de simulación que requieran ser ejecutados en una supercomputadora que tendrá restringido el uso y consumo de recursos a cada autoridad de forma específica. El acceso se controla de igual forma que en las aplicaciones anteriores mientras que el límite y verificación del consumo de recursos del agente y de su autoridad se llevan a cabo por medio de las asignaciones.

21.2.7 Aplicaciones que requieran cambiar la política de seguridad en tiempo de ejecución

Existen aplicaciones que son iniciadas con una política de seguridad predeterminada de acuerdo a las políticas establecidas, sin embargo durante su ejecución es necesario cambiar los permisos otorgados sin reiniciar el sistema debido a que no se debe dejar de ofrecer el servicio que proporcionan o porque son sistemas que no se pueden interrumpir. Veamos algunos casos:

21.2.7.1 Compra/venta de servicios automatizados en tiempo real

Para describir este caso se unirán los casos de la biblioteca digital y el centro comercial electrónico descritos anteriormente. Supongamos que la biblioteca *Allbook* al igual que el centro comercial electrónico *Galerías* tienen instalado el sistema MILENIO y toda su gestión se lleva a cabo mediante agentes móviles.

Allbook lleva a cabo la autenticación y la asignación de permisos de acuerdo a las normas que define MILENIO y basándose en los servicios que le han contratado sus clientes, es decir, el acceso que tendrá el agente móvil sólo será a las materias que éste haya contratado.

Adicionalmente *Allbook* envía un representante a *Galerías* para que sus clientes puedan ir a contratar más materias las 24 horas del día, no obstante la biblioteca garantiza a sus clientes que una vez hecha la negociación con su agente representante, podrán acceder a

los servicios contratados unos momentos después. Este esquema es representado en la figura 21.2.7.1.1.

Como se puede apreciar, la autoridad *Oviedo3* envía a un agente a *Galerías* para que compre el acceso a los libros de electrónica de *Allbook* una vez que ha hecho la negociación, el representante de *Allbook* se lo notifica al gestor de permisos del sistema (MILENIO) quien se encarga de incluir a *Oviedo3* dentro de la lista de los usuarios autorizados, posteriormente invoca el método que se encarga de actualizar la política de seguridad para que los nuevos cambios entren en vigor.

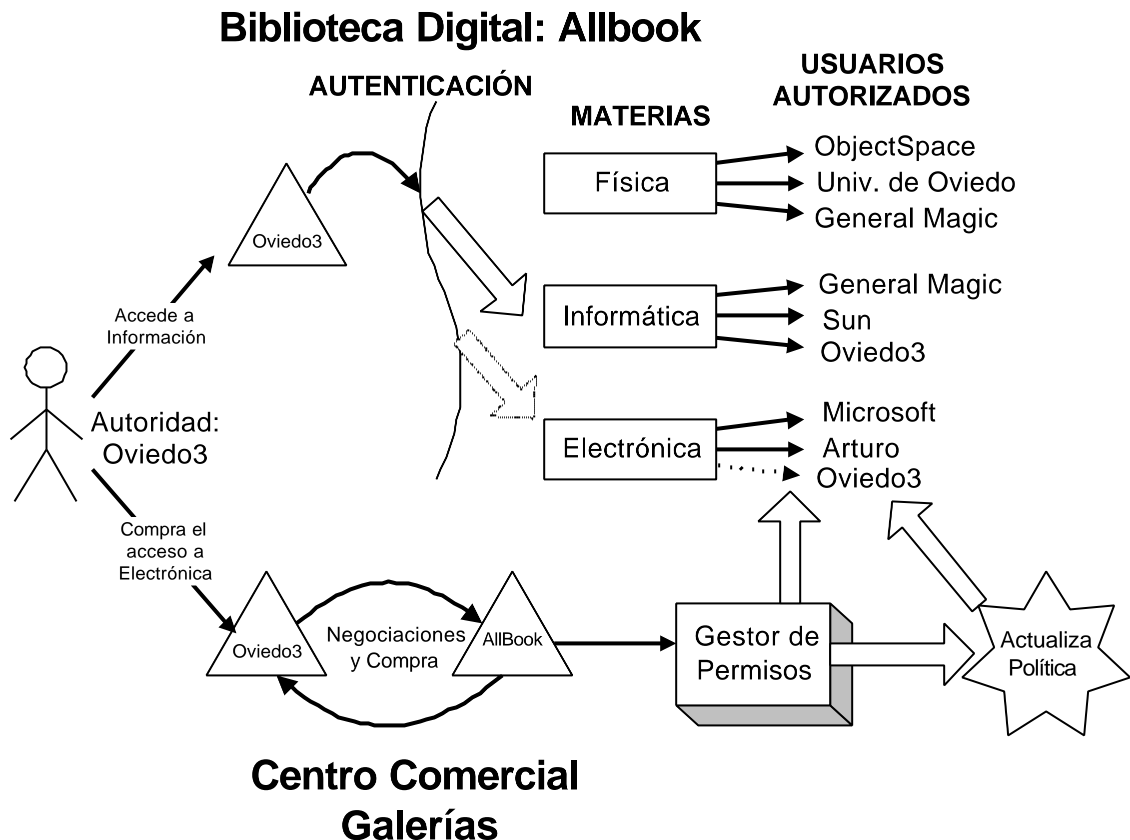


Figura 21.2.7.1.1 – Compra/Venta de Servicios y Actualización de Políticas en Tiempo Real

Una vez modificada la política los agentes de *Oviedo3* podrán acceder también a los libros de electrónica. Es importante señalar que todas estas modificaciones se llevan a cabo casi instantáneamente y son posibles gracias al sistema de seguridad de MILENIO.

21.2.7.2 Actualización de permisos en sistemas que no se pueden interrumpir

La aplicación de MILENIO hacía estos sistemas, al igual que en el ejemplo anterior es debido a que su política de seguridad se puede cambiar y actualizar en tiempo real sin necesidad de reiniciar el sistema.

Existen aplicaciones que por su naturaleza no pueden ser detenidas para ser cambiadas y que requieren que en todo momento que se supervise el cumplimiento de privilegios de

los usuarios que accedieron al sistema en un momento específico de acuerdo a los privilegios que se le dieron ese momento.

Además dichas aplicaciones llegan a requerir que esos privilegios de acceso sean modificados y se actualicen sin afectar el cumplimiento de privilegios de los que ya estaban en el sistema, para esas aplicaciones resulta ideal MILENIO. Entre ellas podríamos indicar:

- Bancos.
- Bolsas de Valores.
- Acceso a sistemas de investigación que cuentan con información confidencial y por consiguiente privilegiada.
- Etc.

Capítulo XXII
CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS

Una vez presentada la arquitectura de seguridad que dio origen a este trabajo, su implementación, su utilidad para otros trabajos relacionados, su evaluación y sus ámbitos de aplicación, se presenta en breve las conclusiones y aportaciones de éste trabajo así como las líneas de investigación futuras.

22.1 CONCLUSIONES

Esta investigación comenzó introduciendo la tecnología de agentes y describiendo tres de las plataformas más importantes para el desarrollo de agentes así como las grandes carencias de seguridad que tienen estos sistemas al igual que la mayor parte de los sistemas de agentes móviles actuales. Se mostró como las carencias de seguridad afectan y entorpecen la evolución de esta tecnología.

Los agentes móviles no han podido aprovechar el gran potencial que poseen debido a la desconfianza generalizada que existe al aceptar la ejecución de código extraño en nuestros ordenadores, debido a ello esta investigación deduce que aunque la tecnología tiene muchas carencias, se deben duplicar esfuerzos para resolver el problema de seguridad ya que aunque otras deficiencias sean solucionadas la tecnología no evolucionará y se extenderá mientras no solucione su problema de seguridad.

Por las razones descritas en el párrafo anterior, esta tesis propone la creación de una arquitectura de seguridad integral para sistemas de agentes móviles (SAHARA) que ofrezca un entorno de ejecución seguro en todo sistema que la implemente y que solucione en gran medida el problema de seguridad que se le ha imputado a la tecnología. La tesis describe detalladamente las medidas necesarias para proteger al servidor de agentes, la transmisión entre servidores y para proteger al agente móvil.

Los capítulos posteriores se dedicaron a verificar que la implementación de la arquitectura de seguridad integral es posible. Para ello se creó un sistema de agentes

móviles con una funcionalidad básica el cual fue llamado MILENIO, sobre este sistema se implemento la arquitectura de seguridad SAHARA para convertirlo en un sistema de agentes móviles seguro. La implementación del sistema siguió detalladamente los objetivos de SAHARA y desarrolló con detalle casi todos los elementos de protección de la arquitectura. Adicionalmente se presentaron las aplicaciones de SAHARA a los mecanismos de seguridad de los sistemas de agentes móviles actuales. Finalmente se muestra la evaluación de MILENIO comparando su sistema de seguridad con el de otros sistemas actuales y los ámbitos de aplicación en los que puede usarse.

El resultado de toda la investigación es un sistema de agentes móviles portable[122] con una **arquitectura de seguridad integral más completa** (que viene a ser la aportación principal de esta tesis)**, más versátil, segura y eficiente**[123], capaz de proteger todos[124] los ataques que un sistema de agentes móviles y sus agentes puedan sufrir, con un entorno visual de ejecución que permite el desarrollo de cualquier tipo de aplicaciones de agentes móviles, incluyendo por supuesto, aquellas que exijan los más grandes requerimientos de seguridad.

En la primera parte de la tesis se ofreció una introducción a las tecnologías de objetos distribuidos que fueron las precursoras de la tecnología de agentes, se observaron algunas de sus tendencias que concluyeron en objetos móviles dando origen a la tecnología.

En la segunda parte se ofreció una introducción a la tecnología, describiendo sus beneficios y posibles aplicaciones, su integración con sistemas orientados a objetos y con la tecnología CORBA. Finalmente se analizaron algunas plataformas de desarrollo actuales concluyendo que su mayor carencia era la seguridad. Se determinaron los requerimientos de un sistema de agentes ideal y se determinaron las necesidades de seguridad que se deberían satisfacer

En la tercera parte se diseñó la arquitectura de seguridad integral SAHARA cuyos objetivos satisfacen los requerimientos de seguridad actuales, en ella se planteo la forma de proteger a un sistema de agentes móviles así como a los agentes de todos los ataques posibles que pueden sufrir. Finalmente se implementó completamente la especificación de la arquitectura de SAHARA para demostrar que es factible y eficaz su implementación.

En la última parte se describió el impacto que puede tener esta investigación en otros trabajos relacionados, se compararon sus características con las de otros sistemas y se definieron sus ámbitos de aplicación posibles.

Los resultados obtenidos en esta tesis pueden aplicarse a otros sistemas de agentes de manera individual de tal forma que al combinarse con los esquemas de seguridad de éstos mejoren su sistema de seguridad respectivo.

---

[122] A todas la plataformas que soporte la JVM.
[123] Que puede ser implementada fácilmente en cualquier otro sistema de agentes móviles.
[124] Excepto ataques al estado de datos del agente.

22.2 RESULTADOS A DESTACAR DENTRO DE LOS DIFERENTES ASPECTOS DE LA ARQUITECTURA DE SEGURIDAD SAHARA

Durante el diseño de la arquitectura de seguridad SAHARA y durante su implementación en el sistema MILENIO como fruto de la investigación y de la experiencia de haberlos implementado se obtuvieron algunas conclusiones o resultados que pueden destacarse y que pueden ser relevantes y de utilidad al aplicarse a otros sistemas. Éstos conceptos se aplicaron a la arquitectura de seguridad SAHARA y al sistema MILENIO y pueden presentarse de forma resumida en los siguientes apartados.

22.2.1 Del Diseño General de SAHARA

22.2.1.1 Diseño de una arquitectura genérica e integral para sistemas de agentes móviles

Se diseño una arquitectura de seguridad integral genérica de referencia para implementar modelos de seguridad para sistemas de agentes móviles que cuenta con protección para el servidor, protección para la transmisión de agentes y su información, protección para los agentes móviles y adicionalmente cuenta con la capacidad de monitorizar y gestionar sus recursos por sí misma (capítulo XI).

22.2.1.2 Uso de lugares para definir grupos afines que compartan recursos

El uso de lugares no debe limitarse a definir contextos de ejecución que agrupe a un conjunto de agentes sino que debe además definir grupos de trabajo afines con privilegios comunes que les permita compartir recursos de forma similar. Igualmente durante la ejecución de los agentes se debe permitir que los agentes del lugar accedan de acuerdo a los privilegios establecidos a los ficheros generados por los agentes del mismo lugar (capítulo XI).

22.2.1.3 Uso de regiones y grupos de regiones para crear redes seguras

Es preciso que para evitar los costes de desempeño tan grandes al introducir todos los mecanismos de seguridad en un sistema se definan regiones, que son grupos de servidores que pertenecen a una sola autoridad y en donde son innecesarias las validaciones de seguridad. De igual forma es posible definir grupos de regiones entre autoridades de un gran nivel de confianza para crear redes seguras y para que los sistemas incluidos en ellas gocen de los beneficios descritos (capítulo XI).

22.2.1.4 Uso de ficheros **Jar** *para transportar agentes más eficazmente*

El uso de ficheros *Jar* para transportar agentes viene a ser una forma innovadora introducida por SAHARA ya que permite primeramente que dentro de un sólo fichero viaje el agente y las clases requeridas para su reanudación. Facilita el proceso de firmado de clases para la protección del agente. Al viajar comprimido acelera su transferencia y permite dentro del mismo fichero crear una jerarquía basada en carpetas en donde se pueden agrupar el estado de datos, el estado de código del agente así como las firmas requeridas para su validación de forma independiente (capítulo XII).

21.2.1.5 Uso de un mecanismo de validación de acceso al sistema

La restricción del acceso al sistema mediante una cuenta de acceso y una clave secreta que posea cada usuario autorizado evita la mala administración o mal uso del sistema. La cuenta de Administrador debe ser única y gozará de todos los privilegios mientras que las de usuarios/autoridades sólo podrán crear y enviar agentes. Para evitar tener dos repositorios de claves secretas, debe de usarse el mismo repositorios donde se guardan las claves privadas de los usuarios del sistema, es decir, las cuentas de acceso al sistema se deben validar con las autoridades que ha sido dadas de alta en el sistema. Siguiendo este esquema sólo el administrador podrá incorporar nuevas autoridades al sistema y ninguna persona no autorizada podrá acceder o hacer mal uso de los recursos (cap. XI).

22.2.2 Protección del Servidor

22.2.2.1 Autenticación individual de autoridades propietarias de agentes

Es indispensable la autenticación particular de todos los propietarios de los agentes para poder otorgar privilegios particulares a los distintos usuarios, tanto locales y remotos que tengan o envíen agentes a un sistema particular (capítulo XII).

22.2.2.2 Especificación de un repositorio de Claves y Certificados encriptado

Para lograr la seguridad de las claves de las autoridades es necesario definir una sintaxis apropiada para guardar las claves privadas de los usuarios locales y los certificados de clave pública de los usuarios remotos. Adicionalmente la información debe mantenerse codificada para evitar cualquier intento de falsificación o mal uso de ella (capítulo XII).

22.2.2.3 Autorización de privilegios individual mediante dominios de protección para cada autoridad o sistema remoto.

El crear un dominio de protección por cada autoridad o sistema remoto favorece la gestión de la asignación de recursos porque una vez creado el dominio cualquier agente adicional de esa autoridad que llegue al sistema simplemente será incluido en el dominio el cual establece los privilegios que va a tener. Adicionalmente el cumplimiento de privilegio será controlado por dicho dominio de protección (cap. XII).

22.2.2.4 Protección de recursos individuales con privilegios de acceso específicos mediante una política de seguridad versátil

El otorgar permisos genéricos sobre un recurso plantea muchos inconvenientes ya que una vez concedido el acceso al recurso el autorizado podría hacer lo que quisiera (Por ejemplo con los archivos del sistema de ficheros). Una política de seguridad versátil debe contar con una sintaxis flexible que permita indicar sobre que recurso especifico se concede el permiso así como sus privilegios de acceso. (Por ejemplo permiso de lectura sobre el fichero C:\autoexec.bat) (capítulo XII).

22.2.2.5 Posibilidad de compartir recursos de forma protegida en tiempo de ejecución

La capacidad de que un agente pueda compartir sus recursos o ficheros en tiempo de ejecución indicando sus privilegios de acceso favorece la seguridad de las aplicaciones que se desarrollan a la vez en grupo y en paralelo (capítulo XII).

22.2.2.6 Control del consumo de recursos por cada agente y por cada autoridad remota

Con el control de los recursos que consume cada agente individualmente y de los que han consumido todos los agentes de esa autoridad se evita el ataque de “agotamiento de recursos”, si sólo se mide lo que consume cada agente, una autoridad podría crear *N* agentes que consuma la cantidad permitida del recurso hasta que llegue al límite del sistema (capítulo XII).

22.2.2.7 Creación de dominios de protección para la asignación de privilegios

Al considerar como unidad base de asignación de permisos al *dominio de protección* se permite que dicha entidad englobe todos los permisos de una autoridad[125] y que a está se le asocien todos lo agentes que pertenezcan a dicha autoridad, evitando con ello

[125] O bien un grupo de ellas que inclusive puede incluir un codebase determinado.

mantener ACL(listas de control de accesos) o tablas de permisos por cada agente repitiendo en gran medida los permisos de los agentes que pertenezcan a una misma autoridad. (capítulo XII y XVII).

22.2.3 Transmisión segura

22.2.3.1 Autenticación de servidores

Para garantizar una transmisión segura es fundamental que el servidor local autentifique la autoridad del servidor remoto antes de abrir una conexión, para que pueda determinar si confía o no en esa autoridad. Al ser autentificado se procederá a abrir la conexión, a definir el protocolo de encriptación y hacer el envío (capítulo XIII).

22.2.3.2 Transmisión de agentes protegida

La posibilidad de espionaje del estado de código y del estado de datos del agente así como la falsificación del mismo se evita utilizando un protocolo de transferencia segura que autentifique al servidor destino y que encripte la información antes de enviarla. SSL satisface dichos requerimientos (capítulo XIII).

22.2.4 Protección del Agente

22.2.4.1 Código del agente protegido mediante firmas digitales

La única forma de garantizar que el agente no ha sido alterado durante su recorrido por un servidor malicioso o por terceras partes es firmando su estado de código con la firma digital de la autoridad a la que pertenece. Si alguien lo modifica, la firma ya no será válida al ser autentificado y será rechazado por el servidor al que intente ingresar (capítulo XIV).

22.2.4.2 Protección del estado de datos mediante diversas técnicas

La protección total del estado de datos es todavía tema investigación ya que no se ha encontrado una técnica que se pueda implementar que solucione el problema[126]. SAHARA propone diversas técnicas para la protección parcial de datos, la más apropiada y eficiente son los PRAC, no obstante se puede firmar el estado de datos con la firma de la autoridad del servidor para adjudicar responsabilidades en caso de daños o perjuicios sobre el agente (capítulo XIV).

[126] Existe la técnica propuesta por Fritz Holh [HOHL97] y la de Tomas Sander[SAND97b] pero no resulta factible su implementación.

22.2.5 Autocontrol del Sistema

22.2.5.1 Modificación de la política de seguridad en tiempo de ejecución

La arquitectura SAHARA permite la modificación de la política de seguridad en tiempo de ejecución para que los sistemas que la implementen puedan ser utilizados por sistemas que no se pueden interrumpir como sistemas bancarios, etc. (capítulo XV).

22.2.5.2 Uso de un mecanismo de control inteligente para mantener la estabilidad del sistema.

La técnica de supervisión y autocontrol de recursos permite que el sistema permanezca siempre estable ya que cuando se detecte alguna escasez no permitirá que se sigan consumiendo recursos y de ser necesario modificará la política de seguridad para evitar otorgar permisos que en ese momento no se puedan satisfacer (capítulo XV).

22.2.6 Del Diseño e Implementación de MILENIO

22.2.6.1 Diseño modular, flexible y ampliable

El sistema ha sido creado modularmente lo que facilita la integración al implementarlo en otros sistemas, de igual forma de igual forma favorece la modificaciones y la ampliaciones (capítulo XVI).

22.2.6.2 Identificadores únicos en tiempo y espacio

Se logran identificadores únicos de longitud fija usando una función de conversión de *M* a *N* y considerando para su generación la dirección IP de origen, el nombre del servidor más la fecha que incluye la hora con detalle en milisegundos del momento en el que fue creado el agente. Estos tres elementos garantizan que no existirán dos identificadores iguales en ningún sistema y en ningún momento durante su ejecución (capítulo XVII).

22.2.6.3 Trasferencia selectiva y eficiente de clases

Se utiliza una técnica para enviar las clases que permite incluir en el fichero Jar (en el que se envía al agente) sólo aquellas clases que serán indispensables para la creación del agente en el sistema remoto omitiendo aquellas que pertenecen a la máquina virtual de Java o aquellas clases que se encuentran dentro de métodos del agente que probablemente nunca serán invocados. Esta técnica mejora aquellas que incluyen todas las clases evitando el coste innecesario de enviar clases que probablemente no se usarán así como aquellas técnicas que solicitan cada clase en demanda ya que se ahorra el

tiempo de iniciar una conexión para cada una de las clases que son indispensables para la reanudación del agente (capítulo XVII).

22.2.6.4 Aceleración del envío de agentes utilizando técnicas de compresión

El tiempo de envío de los agentes, sus clases y su estado de datos, disminuye considerablemente debido a que todos estos componentes viajan comprimidos dentro de un fichero Jar (Capítulo XVII).

22.2.6.5 Uso de itinerarios modificables en tiempo de ejecución para definir el recorrido de los agentes

Mediante los itinerarios es posible definir recorridos frecuentes que pueden ser usados por uno o más agentes, además éstos tienen la capacidad de modificarlo, reducirlo o incrementarlo de acuerdo a su programación, lo que favorece la realización de tareas en grupo así como la programación de agentes inteligentes quienes basándose en sus reglas de inferencia pueden seguir un determinado recorrido.

22.2.6.6 Uso de una interfaz gráfica de alto nivel para la gestión y monitorización del sistema de agentes y para la creación de los mismos.

Se creo una interfaz gráfica provee un medio versátil y eficiente al administrador de llevar a cabo todas las tareas de gestión del sistema desde la creación de nuevos usuarios/autoridades hasta la configuración del sistema de seguridad, de igual forma permite a todos los usuarios la fácil creación y monitorización de agentes (Capítulo XIX).

22.2.7 Aplicación de los resultados de SAHARA a otros Sistemas de agentes

22.2.7.1 Mejora al modelo de seguridad de los aglets

El sistema de seguridad de los aglets puede mejorar su eficiencia creando dominios de protección en vez de hacer uso de un interprete con lo que logra mayor rapidez en la asignación de permisos, puede reducir complejidad en la definición de permisos adoptando el número de autoridades que considera SAHARA y se hace más robusto al contabilizar la cantidad de recursos que consume cada autoridad (siguiendo el modelo de SAHARA) evitando con ello sufrir un ataque de agotamiento de recursos (Capítulo XX).

22.2.7.2 Mejoras a la seguridad de los sistemas de agentes móviles en general

La utilización de algunas ideas propuestas en la arquitectura de seguridad SAHARA permite mejorar los modelos de seguridad de varios sistemas de agentes móviles para que ofrezcan un modelo de seguridad integral, algunas ideas que pueden favorecer dichos modelos son: compresión y modularización de agentes, restricción de acceso al sistema, política de seguridad dinámica, creación de redes confiables mediante el concepto de región y firmas digitales por parte del servidor emisor sobre el estado de datos para adjudicar responsabilidad (Capítulo XX).

22.2.8 Evaluación de MILENIO

22.2.8.1 MILENIO es más versátil y cuenta con mayor número de capacidades de seguridad que los sistemas de agentes actuales.

Considerando las trece características de seguridad más importantes que un sistema de agentes móviles debe tener, MILENIO cuenta con doce de ellas[127] lo que lo convierte en el sistema más seguro de la actualidad seguido por Concordia que cuenta con cinco y una implementada parcialmente. Adicionalmente el sistema es mucho más versátil ya que permite modificar la política de seguridad en tiempo de ejecución (Capítulo XXI).

22.2.8.2 MILENIO cuenta con un desempeño aceptable al ofrecer una arquitectura de seguridad completa.

Los sistemas al ser más seguros suelen tener una perdida de desempeño considerable. No obstante, en pruebas de desempeño realizadas con los aglets que era el sistema con menos características de seguridad, MILENIO resultó ser el doble de lento en las tareas de envío[128], y aproximadamente igual de rápido en tareas de creación y reanudación de agentes. Por lo cual el sistema cuenta con un desempeño aceptable considerando todas las capacidades de seguridad que ofrece y que la velocidad no suele ser la característica más importante de los sistemas de agentes móviles (Capítulo XXI).

22.2.9 Aplicaciones de MILENIO

22.2.9.1 Posibilidad de aplicación flexible a un gran rango de aplicaciones

La flexibilidad del sistema es tal que puede usarse en un gran rango de ámbitos[129], que pueden ir desde distribución selectiva y certificada de información (que pueden ser

[127] Dos de ellas implementadas parcialmente, pero con la infraestructura ya definida para añadir las validaciones en cuanto lo permita la JVM.
[128] Debido a las firmas digitales de los agentes, sin embargo si se enviaban sin firmar MILENIO era más rápido.
[129] Por ejemplo para todas las aplicaciones típicas de agentes móviles que son descritas en el capítulo VII.

programas o correo electrónico) hasta aplicaciones concurrentes como pueden ser la simulación de una supercomputadora o el balanceo del tráfico en grandes redes (Capítulo XXI).

22.2.9.2 Ofrece nuevas oportunidades para el comercio electrónico.

La fiabilidad y seguridad en las transacciones permite crear nuevos patrones para el comercio electrónico[130] que beneficia a las aplicaciones en donde se paga el acceso individual a un recurso (como en bibliotecas digitales) y a aplicaciones de compra ventas de productos/servicios en donde los agentes podrán negociar para obtener el mejor precio e inclusive realizar la consumación de operaciones efectuando el pago respectivo (Capítulo XXI).

22.2.9.3 Uso en aplicaciones que requieran cambiar su política de seguridad en tiempo de ejecución.

MILENIO puede utilizarse para aplicaciones que no se pueden interrumpir donde sea necesario modificar la política de seguridad (para añadir o quitar privilegios de acceso) en tiempo de ejecución sin reiniciar el sistema como pueden ser bibliotecas digitales[131], aplicaciones bancarias, bolsas de valores y sistemas que controlan información confidencial (Capítulo XXI).

22.2.3 LINEAS DE INVESTIGACIÓN FUTURAS

Nuestro trabajo principal se centra en refinar pequeñas tareas pendientes para concluir una arquitectura de seguridad integral y total, por ejemplo el añadir todo un modulo de seguridad para el intercambio seguro de mensajes entre agentes locales y remotos. Mi siguiente objetivo será enfocado a la interoperabilidad de los sistemas de agentes, es decir, lograr que nuestro sistema de agentes móviles pueda interactuar con agentes móviles de otros sistemas y viceversa pero conservando la seguridad en las operaciones.

22.2.3.1 Mecanismo de distribución de claves públicas entre servidores

22.2.3.2 Protección de datos del agente.

[130] Al grado de poder definir en centro comercial electrónico.
[131] Que deseen incorporar el acceso a sus nuevos clientes sin dejar de dar el servicio a los otros.

22.2.3.3 Intercambio de mensajes e información entre agentes de forma segura

22.2.3.4 Interoperabilidad con otros sistemas de agentes móviles

FUENTES DE INFORMACION

BIBLIOGRAFIA:

[ACEB99] Acebal César F., Pérez Díaz Jesús Arturo, Cueva Lovelle Juan Manuel. *Remote database querying using mobile agents.* Publicado en el "World Multiconference on Systemics, Cybernetics and Informatics and 4th International Conference on Information Systems, Analysis and Synthesis". ISBN*:* 980-07-5914-X. Orlando, USA. July, 1999

[ALVA98] Alvarez Gutierrez Dario. *Tesis Doctoral - Persistencia completa para un sistema operativo orientado a objetos usando una máquina abstracta con arquitectura reflectiva.* Enero 1998, Departamento de Informática – Universidad de Oviedo.

[ARID98] Aridor Yariv, Lange Danny. *Agents Design Patters: Elements of Agent Application Design*. Proccedings of the second international Conference in autonomous Agents in Minnepolis, Mayo 1998.

[ARID98b] Aridor Yariv, Oshima Mitsuru. *Infraestructure for Mobile Agents: requierements and design.* Proceedings id the 2nd international worshop on mobile agents, September 1998. Alemania.

[BERK98] Berkovis S., Guttman J.D., Swarup V. *Authentication for Mobile Agents.* The MITRE Corporation. 1998.

[BROW96] Brown Nat, Kindel Charlie. *Distributed Component Object Model Protocol -- DCOM/1.0.* Microsoft Corporation. November, 1996 http://www.microsoft.com/workshop/prog/com/

[BROW95] Browne Shirley. *Need for a Secure Profile for Agent Execution Environments.* Proceeding of hte CIKM Worshop on Intelligent Information Agents. 1995

[CHES95] Chess David, Harrison Colin. *Mobile Agents: are They a good idea?.* IBM Reaserch Division. New York, USA 1995.

http://www.research.ibm.com/massive.

[CHES97] Cheswick Willian R. *Internet Security and FireWalls:Repelling the Wily hacker*. Editorial Addison Wesley, 1997.

[COCK98] Cockayne Willian R., Zyda Michael. *Mobile Agents.* Editorial Manning Publications. U.S.A. 1998. CAP 2,3,4 y 5. ISBN 1-884777-36-8.

[CUEV98a] Cueva Lovelle Juan Manuel. *Análisis y diseño orientado a objetos*. Cuaderno didáctico, Universidad de Oviedo, Marzo 1998. Pp 6-29.

[CUEV98b] Cueva Lovelle Juan Manuel, Pérez Díaz J. Arturo, López Pérez Bejamín. *Interoperabilidad de Sistemas de Información Geográfica utilizando máquinas virtuales, componentes y agentes móviles.* Memorias del Seminario Internacional Sobre sistemas de Información Geográfica. Colombia, Mayo 1998.

[DAGE96] Dageforde María. *Security in the JDK 1.1*. Sun Microsystems. San Antonio, U.S.A. 1996.

[DEAN97] Dean Drew and Felten Edward W. Secure Mobile Code: ¿ Where do we go from here?. Dept of Computer Science. Princeton University. Princeton, NJ08544. Publicado en el Workshop on Formdations for Secure Mobile Code, Monterey, CA, March 26-28, 1997.

[DORI97] Dorigo Marco, Gambardella Luca Maria. *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.* IEE 1997.

[FARL97] Farley Steven R. *Mobile Agent System Architecture: A flexible alternative to moving data and code to complete a given task.* SIGS Publications, 1997.

[FARM98] Farmer William M, Guttman Joshua D, and Swarup Vipin.*Security for Mobile Agents: Issues and Requeriments.* The MITRE Corporation. 1998.

[FISC99] Fischmeister and Wolfang Lugmayr. *The supervisor-Worker Pattern.* Techical University of Vienna. Information Systems Institute. Distributed Systems Group. Mayo, 1999.

[FINI95] Finin Tim, Labrou Yanis, Mayfield Jarnes. *KQML as an agent comunication lenguaje.* Septiembre 1995.

[FREI96] Freier Alan, Karlton Philip, Kocher Paul. *The SSL Protocol Version 3.0.* Transport Layer Security Working Group, Netscape Communications. Noviembre, 1996.

[GRAY95a] Gray Robert S. *Agent Tcl: Alpha realease 1.1.* Dartmouth College, Hanover. 1995. Pp 17-35.

http://www.cs.dartmouth.edu/~agent/agent2.0/download.html

[GRAY95b] Gray Robert S. *Agent Tcl: A transportable agent system*. In Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland, December 1995.

[GRAY96] Gray Robert S. *Agent Tcl: a flexible and secure mobile agent system.* Publicado en los Proceedings of the Fourth Annual Usenix Tcl/Tk Workshop. 1996.
http://www.cs.dartmouth.edu/~agent/papers/index.html

[GRAY97] Gray Robert S. *Agent Tcl: a flexible and secure mobile agent system.* Tesis Doctoral, Darmouth college, Hanover 1997.

[GONT96] Gont Li. *New Segurity Architectural Directions for Java.* JavaSoft, Cupertino, California. December, 1996

[GONT98] Gont Li and Schemers Roland. *Implementing protection Domains in the Java TM Developemt Kit 1.2.* JavaSoft, Sun Microsystems, Inc. March 1998.

[GROS95] Grosling, J. And McGilton, H. *The Java lenguaje environment: a white paper.* Technical report, Sun Microsystems, 1995.

[HIDA99] Hidalgo Martín Renato. *Millennium: Sistema de Agentes Móviles.* Proyecto de Fin de carrera, EUITIO, Universidad de Oviedo, Septiembre de 1999.

[HOGI97] Hagimont Daniel, Iamail Leila. *A protection scheme for Mobile Agents on Java.* Instituto Nacional de Investigación en Informática y automatización. Saint-Martin, Francia. 1997.

[HOHL96] Hohl Fritz, Baumann Joachim and Straßer Markus. *Beyond Java: Merging Corba-based Mobile Agents and WWW.* Joint W3C/OMG Workshop on Distributed Objects and Mobile Code, June 1996 Boston, Massachusetts.
http://www.w3.org/OOP/9606_Workshop/submissions/20-pospaper3.html

[HOHL97] Hohl Fritz. *An approach to solve the problem of malicious hosts in mobile agent systems.* Institute of parallel and distributed systems, University of Stuttgart, Germany. 1997.

[HOHL98] Hohl Fritz, Peter Klar, Joachim Baumann. *Efficient code migration for modular mobile agents.* Institute of parallel and distributed high performance systems, University of Stuttgart, Germany. 1998.

[HUNT94] Hunt Craig, *TCP/IP Network Administration.* Editorial O'Really & Asociates. ISBN 0-937175-82-X.

[HURS97] Hurst Leon, Cunningham Pádraig. *Mobile Agents – Smart Messages*. Proceedings of the First International Workshop on Mobile Agents, MA'97, Berlin Germany.

[IBM95] Gilbert, Aparicio, et al. *The Role of Intelligent Agents in the Information Infrastructure*. IBM, United States, 1995. http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm

[KARJ97] Karjoth Guenter, Oshima Mitsuru, Lange Danny. *A security Model for Aglets.* IBM Research division. Julio-Agosto 1997.

[KARJ98] Karjoth Guenter.

[KAUF95] Kaufman C., Perlman R., Spicener M. *Network Security: Private Communication in a Public World*. Prentice Hall, 1995.

[JAWO99] Jaworski Jamie. *Java 1.2 al descubierto.* Prentice Hall. ISBN: 1-57521-369-3. 1999.

[LING96] Lingnau Anselm, Drobnik Oswald. An infrastructure for mobile agents: requierements and arquitecture. JW Goethe University. Frankfurt, Alemania. 1996.

[LANG96] Lange Danny, Chang Daniel T. *IBM Aglets Workbench Programming Mobile Agents in Java A White Paper Draft.* IBM Corporation. Septiembre 1996.

[LANG98] Lange Danny. *Mobile Agents: Enviroments, Technologies, and applications*. Proceedings of the Practical Application of intelligent Agents and Multi-Agent Technology Conference, PAAM98. Marzo 1998.

[LANG98b] Danny Lange, Oshima Mitsuru. Programming and Deploying Java Mobile Agents with Aglets. Ed. Addison Wesley. ISBN 0-201-32582-9. U.S.A, 1998.

[LAWT96] Lawton George. *Agents to roam the Internet.* Sunworld Online Magazine. Octubre de 1996. http://www.sun.com/sunworldonline/swol-10-1996/swol-10-agent.html

[MEAD97] Meadows Catherine. *Detecting Attacks on Mobile Agents.* Center for High Assurance Computing Systems. Naval Research Laboratory. Washington, 1997.

[MOOR98] Moore T. Jonathan. *Mobile Code Security Techniques*. Department of Computer and Information Science. University of Pennsylvania. 1998.

[MORR98] Morrison Michel. *Java Unleashed. Second Edition.*Sams.Net. ISBN.1-57521-197-1. 1998.

[MICR96] Microsoft Corporation. *DCOM Technical Overview*. USA, 1996. http://www.microsoft.com/workshop/prog/com/

[MURU98] Murugesan San. *Intelligent agents on the internet and web: progress and prospects.* Proceedings of the International Workshop on Intelligent Agents on the Internet and Web. March 1998, Mexico. pp 3-6.

[NAUG97] Naughton Patrick, Herbert Shildt. *Java - Manual de referencia*. Editorial Mc Graw Hill. España 1997

[NISH95a] Nishigaya Takashi, *Design of MultiAgent Programming Libraries for Java.* Fujitsu Laboratories. Japan 1995.

[NISH95b] Nishigaya Takashi, *Agent Security Model.* Fujitsu Laboratories. Kawasaki, Japan 1995.

[OBJE97] ObjecSpace Voyager. *ORB 3.0 Developers Guide*. Primera Edición. USA 1997-99.

[OMG97] OMG, Crystaliz, General Magic, GMD Fokus, and IBM. *Mobile Agent System Interoperability Facilities Specification*. Noviembre 10, 1997.

[ORDI96] Ordille Joann J. *When agents roam, who can you trust?.* Computer Science Research Center. NJ, USA, 1996.

[ORFA96] Orfalli Robert, Harkey Dan, Edwards Jeri. *The essential Distributed Objects survival Guide.* Editorial John wiley & Sons, Inc. 1996. Cap 1-10,20-29.

[OSHI97] Oshima Mitsuru, Karjoth Guenter. *Aglets Specification (Alpha5) Draft.* Draft 2.0. IBM Japón. Septiembre 1997. http://www.trl.ibm.co.jp/aglets/documentation.html

[OUST94] Ousterhout, J. *TCL and Tk Toolkit.* AdisonWesley, Reading, MA. 1994

[PEIN97a] Peiner Holger. *An introduction to mobile agent programming and the Ara system.* University of Kaiserlautern, Alemania. Enero 1997. http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/documents.html

[PEIN97b] Peiner Holger, Stolpmann Torsten. *The Arquitecture of the Ara Platform for Mobile Agents.* Publicado en los Proceedings of the first International Worshop on Mobile Agents, MA'97. Berlin, Germany. Abril 1997

[PEIN98] Peiner Holger. *Security concepts and implementation in the Ara Mobile Agent system.* Publicado en los anales del WETICE '98. Palo Alto California, USA, 1998.

[PERE98a] Pérez Díaz Jesús Arturo, Cueva Lovelle Juan M., López Pérez

Benjamín. *Different Platforms for mobile agents development.* Proceedings of the International Workshop on Intelligent Agents on the Internet and Web. Marzo 1998, México.

[PERE98b] Pérez Díaz Jesús Arturo, Cueva Lovelle Juan M., López Pérez Benjamín. *Mobile agents development on the internet, is java or CORBA the dominant trend?.* Proceedings of the Practical Application of intelligent Agents and Multi-Agent Technology Conference, PAAM98. Marzo 1998, Londres.

[PERE98e] **PARALLEL PROGRAMMING ON THE INTERNET USING MOBILE AGENTS**
Autores: Jesús Arturo Pérez Díaz, Juan Manuel Cueva Lovelle, Benjamin López y Sara Isabel García Barón.
Congreso: "**Second IASTED International Conference "European Parallel and Distributed Systems EURO-PDS'98**".
Lugar: Viena, Austria.
Fecha: Julio de 1998

[PERE98f] Pérez Díaz Jesús Arturo y Cueva Lovelle Juan Manuel, López Pérez Benjamín, Domínguez Mateos Francisco, Luengo Diez Candida, Pérez Francisco Javier, García Baron Sara Isabel. *Desing patterns for mobile agents applications* . V Congreso Internacional de Investigación en Ciencias Computacionales CIICC´98. Noviembre de 1998. Aguascalientes, México

[PEREb98a] Pérez Glz. Francisco Javier, Pérez Díaz Jesús Arturo y Cueva Lovelle Juan Manuel. *Study of the performance of a unix serverwith aglets.* Worshop on European Scientific and Industrial Collaboration on promoting Advanced Tecnologies in Manufacturing, WESIC'98. ISBN 84-95138-08-5. Gerona, España. Junio de 1998.

[PEREb98b] Pérez Glz Francisco Javier, Pérez Díaz Jesús Arturo, Juan M. Cueva Lovelle, Francisco Domínguez Mateos, Candida luengo Diez. A *network monitoring system with aglets.* Quinto Congreso Internacional de Investigación en Ciencias Computacionales CIICC´98.
ISBN: 970-13-2261-4. Noviembre de 1998. Aguascalientes, México

[RETE97] Retelewski Tom. *Create New Event Types in Java: Learn how to make even classes for components in JDK 1.1.* Java Word. August, 1997.

[SAND97a] Sander Thomas, Tshudin Christian. *On Cryptographic Protection of Mobile Agents.* Publicado en las memorias del workshop internacional: Mobile Agent and Security. Baltimore, Octubre de 1997.

[SAND97b] Sander Thomas, Tshudin Christian. *Protecting Mobile Agents against Malicious Host.* Publicado en las memorias del workshop internacional Mobile Agent Security. Noviembre de 1997.

[SAND97c] Sander Thomas, Tshudin Christian. *Toward mobile cryptography.*

International Computer Science Instite. Noviembre de 1997.

[SCHN96] Schneier Bruce. *Aplied Criptography.* Editorial John Wiley & Sons, 1996.

[SEBE89] Seberry Jennifer, Pieprzyk Josef. *CHYPTOGRAPHY: An introduction to Computer Security.* Editorial Prentice Hall, Australia 1989.

[SOMM97] Sommers Bret. *Agents: Not just for Bond anymore.* JavaWorld - Agents Magazine. Abril 1997. Pp 3-6. http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents.html

[STAL95] Stalling William. *Network and Internetwork Security.* IEEE Press, 1995.

[TARD96] Tardo J., Valente L., *Mobile Agent Security and Telescript*. In IEEE CompCon, 1996. http://www.cs.umbc.edu/agents/security/

[TSCH97] Tschuding Christian F. *Mobile Agent Security.* Department of Computer Systems Uppsala University. Suecia. 1997.

[VENN97] Venners Bill. *Solve real problems with aglets, a type of mobile agent*. JavaWorld - Under the hood Magazine. Mayo 1997. Pp 2-4. http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood.html

[VENN97] Venners Bill. *Find out about the inner Workings of aglets, IBM Japan's Java-based autonomous Software agent technology.* Under the hood Magazine. Abril.1997. http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html

[VING97] Vigna G. *Protecting mobile agents throuhg tracing.* In proceedings of the Third Worshop on Mobile Object Systems, Filandia 1997.

[WALL90] Wall L., Schwartz R. *Programming Perl.* O'really & Associates, Sebastopol, CA. 1990.

[WALS98] Walsh Tom, Paciorek Noemi, Wong David. *Security and Reliability in Concordia*. Mitsubishi Electric ITA, Horizon System Laboratory. USA, 1998.

[WEBE97] Weber Joe, Baker David. *Using Java 1.1, The most complete reference*. Tercera edición. Editorial QUE. U.S.A. 1997.

[WHIT96] White Jim. *Mobile Agent white paper*. General Magic, 1996. http://www.genmagic.com/agents/Whitepaper/whitepaper.html, http://www.genmagic.com/technology/techwhitepaper.html

[WILH98] Wilhelm Uwe G., Staamann Sebastian. *Protecting the Itinerary of Mobile Agents.* Anales del ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems. Julio 21-22 1998. Bélgica.

[WOOL95] Wooldridge, M., and Jennings, N.R. *Intelillent Agents: Theory and Practice*. Enero 1995.
http://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-htm.html

[YEE97] Yee Bennet S. *A Santuary for Mobile Agents*. USCD. April 1997.

FUENTES DE INFORMACION EN INTERNET:

[AGLE] Aglets workbench Tutorial. *A Journal Of ideas, opinions and practices collected from the Aglet Mailing list.*
http://www.geocities.com/SiliconValley/Lakes/8849/agletron.htm

[ASTR] Astra Ad Engineering. *Mobilize the enterprise.*
http://www.jumpingbeans.com/

[DARTa] Dartmouth Collegue. *D'Agents Source Code and Documentation.*
http://www.cs.dartmouth.edu/~agent/agent2.0

[KAISa] University of Kaiserslautern. *The Ara platform for mobile Agents.*
http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html

[KAISb] University of Kaiserslautern. *Adaption of a Given Interpreter to the Ara Core.*
http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/Doc/interpreter-adaption.html

[OMG97] OMG. Diagram of the *Mobile Agent System Interoperability Facilities Architecture.*
http://www.agent.org/pub/satp/papers/maf-assessmentA.html

[NETWa] Netware Asociates. *PGP Total Network Security Suite*
http://www.nai.com/products/security/commercial.asp

[SUNa] Sun Microsystems. *Security and Signed Applets.*
http://java.sun.com/products/jdk/1.1/docs/guide/security/index.html

[UBMCa] UMBC AgentWeb. *Knowledge Sharing Effort.*
http://www.cs.umbc.edu/agents/kse/

[UBMCb] UMBC AgentWeb. *Software agents mailing list.*
http://www.cs.umbc.edu/agentslist/archive/current/

A

Acceso a Bases de Datos
Mediante Sistemas de Agentes Basados en Java 217
Agentes Móviles 103
Características 103
Conceptos 106
Definición 103
El paradigma 104
Agentes móviles y otras tecnologías 111
Agentes móviles vs Applets 112
Objetos distribuidos 111
RMI 111
Software intermedio orientado a mensajes 111
Agentes Móviles y Sistemas Orientados a Objetos 208
Clasificación de Patrones de Diseño 212
Incorportación a Sistemas con Tecnología CORBA 219
Integración 209
Representación en UML 210
Aglets 154
Arquitectura 165
Construcción de aplicaciones 174
El modelo del objeto 155
Evaluación 177
Migración de objetos y mensajes 162
Seguridad 172
Albedrío, inteligencia y movilidad de agentes .97
Ámbitos de Aplicación de Milenio 398
Administración de Servidores y Redes 403
Aplicaciones de Pago por Acceso a un Recurso 398
Aplicaciones Diversas que exijan Seguridad 404
Aplicaciones que Requieran Cambiar la Política de Seguridad en Tiempo de Ejecución 405
Comercio Electrónico 399
Distribución Selectiva y Segura de Información 402
Operaciones Bancarias 400
Aplicación de SAHARA a otros Sistemas de Agentes Móviles 375
Al Modelo de Seguridad de los Aglets 380
Al Modelo de Seguridad de Mole 385
Aplicación a otros Sistemas en General 386
Aplicaciones de los agentes móviles 116
Aproximación de Fritz Hohl para Resolver el Problema del Servidor Malicioso 383
Revoltura de Código 384
Tiempo de Vida Limitado de Código y Datos 384
Ara 141
Arquitectura 145
Conceptos de programación 149
Evaluación 153
Generalidades 142
Seguridad 151
Ataques contra Sistemas de Agentes 242
Autoridad 244

C

Carencias de los sistemas de agentes móviles actuales 204
Certificado 244
Clasificación de agentes 98
Diversos tipos de agentes 99
En cuanto a su característica fundamental .. 98
Los agentes en base a su movilidad 98
Cliente/Servidor 19
Con Bases de Datos SQL 19
Con Groupware 21
Con monitores TP 20
Con Objetos Distribuidos 22
Requerimientos del software 19
COM 72
IDLs 72
interfaces de COM 73
objeto COM 73
ODL 72
Servicios de objetos 75
servidor COM 74
Comparación de los sistemas de agentes móviles actuales 203
Comparación de Milenio con otros Sistemas 390
Análisis Comparativo de Seguridad 391
Características Necesarias para ser un Sistema Seguro 392
El Precio de la Seguridad 395
Evaluación del desempeño de MILENIO comparado con los Aglets 396
La Seguridad de Milenio en Comparación con la de otros Sistemas 393
Componentes 23
Características básicas de 24
Consultas a SIG mediante agentes móviles 119
CORBA 26
¿Cómo un componente encuentra su ORB ? 33
El adaptador de objetos 31
El repositorio de interfaces 35
Invocación de métodos 29
Las IDLs y su estructura 34
Los metadatos 34

D

D'Agents 130
Arquitectura 133
Evaluación 140
Generalidades 131
Lenguaje de desarrollo 134
Objetivos de la arquitectura 132
Seguridad .... *Véase*
DCOM 89
Definición de agente 95

E

El Modelo de Seguridad de los Aglets 375
El lenguaje de Autorización........................ 376
La Base de Datos de Políticas del Administrador del Contexto.................. 379
Los Privilegios... 378
Preferencias del Propietario de los Aglets 379
Encriptación 244
Evolución de los agentes 100

F

Facilidad de Interoperabilidad de sistemas de agentes móviles de CORBA (MASIF)...... 183
Conceptos básicos 185
Funciones de un sistema de agentes 189
IDL... 194
Interacción de agentes 189
Interoperabilidad.. 183
La interfaz MAFAgentSystem.................... 198
La interfaz MAFFinder............................... 201
Facilidades Comunes de CORBA 57
Facilidades horizontales.................................57
Facilidades verticales 59
Figuras
Figura 11.2.1 – Interoperabilidad de sistemas de agentes con SAHARA....................... 247
Figura 11.2.3.1 –Migraciones dentro y fuera de una región... 250
Figura 11.2.5.1 – Esquema global de interacción de SAHARA 254
Figura 11.3.2.2.1 – Firmado y Validación de información mediante firmas digitales. 258
Figura 11.3.4.1.1 – Dominios de protección y permisos.. 281
Figura 11.4.3.3.1 – Dominios de protección y Asignaciones 285
Figura 11.5.2.4.1 – Generación de códigos de autenticación de resultados parciales ... 300
Figura 16.2.1 – Marco de trabajo de MILENIO.. 314
Figura 16.2.2 – Componentes de MILENIO 316
Figura 16.2.4.1.1 – Diagrama General de Clases del Núcleo y del Sistema de Agentes 321
Figura 16.2.4.2.1 – Diagrama General de Clases de la Arquitectura de Seguridad 324
Figura 17.1.1 – Estructura del Objeto Agente 328
Figura 17.4.1 – Componentes del Agente al Migrar... 333
Figura 17.6.1 – Clases que intervienen en el Envío/Recepción de un Agente............ 338
Figura 17.6.2 – Protocolo de Transferencia de Agentes 340
Figura 3.1 - La Arquitectura de CORBA 26
Figura 3.1.2 - Invocaciones estáticas y dinámicas de métodos 29
Figura 3.3.1 - Servicios de CORBA : Consulta y relaciones 48
Figura 3.3.1 - Servicios de CORBA : Gestión del sistema y seguridad............................ 52
Figura 3.3.1 - Servicios de CORBA : Nombrado, eventos y ciclo de vida......... 38
Figura 3.3.1 - Servicios de CORBA : Persistencia y bases de datos OO 44
Figura 3.3.1 - Servicios de CORBA : Transacciones y concurrencia 41
Figura 3.9.1 - Facilidades comunes de CORBA.. 57
Figura 4.1 - El modelo de objetos de OpenDoc... 61
Figura 5.1 – Elementos de COM 72
Figura 5.2 – Arquitectura General de DCOM 91
Figura 6.5.1.1 - Aproximación mediante el uso de RPC..104
Figura 6.5.1.2 - Nueva aproximación mediante programación móvil..............105
Figura 6.5.2.1 - Representación de lugares y agentes 106
Figura 6.5.2.2 - Viajes y entrevistas 107
Figura 6.5.3.1 - La máquina y los niveles de los protocolos 110
Figura 7.1.1 - Metodologías de Objetos Distribuidos...112
Figura 7.2.1 - Incremento del número de usuarios en Internet 114
Figura 7.3.1 - Autopista Asturias - Madrid 120
Figura 7.3.2 - Servidores de recopilación parcial de información............................121
Figura 7.3.3 - Delegación de tareas entre agentes 122
Figura 8.3.3.1 - La arquitectura de D'Agents 134
Figura 8.4.2.1 - Vista de alto nivel de la arquitectura del sistema de Ara.............146
Figura 8.4.2.2 - Adaptación del interprete de un lenguaje al núcleo de Ara............149
Figura 8.5.1.1 - Interfaces y clases principales de la API de los Aglets.......156
Figura 8.5.1.2 - Ciclo de vida de un Aglet 161
Figura 8.5.3.1 - Arquitectura de los Aglets 166
Figura 8.5.3.2 - El protocolo de transferencia ATP 167
Figura 8.5.3.3 - Asignación de hilos en el protocolo ATP...168
Figura 8.6.1 - Diversos protocolos de comunicación básicos soportados por KQML..181
Figura 8.7.2.1 - Interconexión entre sistemas de agentes 187
Figura 8.7.2.2 - Interconexión entre regiones 188
Figura 8.7.5.1 - Relación entre la MASIF y un ORB 194

Figura 9.1.1 – Agregación de un Componente de Agentes Móviles para hacer Consultas Remotas 209
Figura 9.1.2 – Agregación de un componente de agentes móviles para vender productos en un mercado electrónico 210
Figura 9.2.1 – Representación de Agentes Locales en UML....... 211
Figura 9.2.2 – Representación de Agentes Remotos en UML....... 211
Figura 9.2.3 – Representación de Agentes Clónicos en UML....... 212
Figura 9.3.1.1 – Patrón de Itinerario 213
Figura 9.3.2.1 – Patrón Maestro-Esclavo . 215
Figura 9.3.3.1 – Patrón de Reunión 216
Figura 9.4.1 – Interacción de Agentes Móviles en Sistemas basados en Java.. 219
Figura 9.5.1 – La Facilidad de Agentes Móviles de CORBA 219
Figura 9.5.2 – Interacción de Agentes Móviles en CORBA 220
Firma Digital....... 244

I

Introducción a los agentes 93

K

KeyStore 362
KQML....... 178
Aplicaciones 182
Descripción 179
Evaluación....... 181
Facilitadores....... 181

L

LLave Privada....... 244
LLave Pública 244

M

Máquina virtual....... 124
MILENIO
Arquitectura de Seguriad SAHARA....... 318
Asignación de Privilegios
Asociando a Dominios de Protección .. 352
Creando Dominios de Protección 353
Autenticación del Agente....... 351
Autocontrol del Sistema y Redefinición de la Política de Seguridad en Tiempo de Ejecución....... 360
Carga Segura del Agente....... 351
Componentes de Administración 317
Compresión del Agente para su Envío 347
Creación e Iniciación de Agentes 330
Cumplimiento de Privilegios 356
Descripción General....... 314
Diagramas Generales de Clases 319
Arquitectura de Seguridad 323
Núcleo del Sistema 320
El Núcleo....... 316
Elementos de un Agente Móvil 327
Firmado Digital de un Agente Móvil....... 345
Generación de Firmas para Autoridades/Usuarios....... 346
Justifiación del Lenguaje Java2 SDK....... 313
Migración de Agentes 332
Necesidades del Lenguaje....... 313
Permisos Otorgados por Agentes 359
Protección de los Datos del Agente....... 348
Protocolo de Transferencia de Agentes 337
Seguridad del Repositorio de Claves y de los Ficheros de Seguridad....... 343
Seguridad en el Acceso al Sistema y en la Asignación de Recursos 341
Seguridad en el Envío y en la Transmisión 348
Serialización de Agentes 331
Transferencia de Clases 334
Desde el Lugar de Procedencia 336
Enviadas con el Agente 335
Verificación de Asignaciones 354

O

Objetos de Negocios....... 56
La anatomía 56
Objetos Distribuidos 23
OLE....... 77
Almacenamiento estructurado y monikers.. 83
Automatización, Programación y librerías de tipos....... 77
OCXs y Documentos compuestos....... 86
Transferencia de datos uniforme....... 81
OpenDoc 61
Automatización y eventos semánticos 68
Bento y unidades de almacenamiento....... 66
El modelo de documentos compuestos 64
El modelo del objeto....... 61
Transferencia de datos uniforme....... 67
ORB CORBA 26
El ORB mundial y la arquitectura Inter-ORB 33
La anatomía 27

P

Patrones de Diseño
De Interacción....... 215
De Migración 212
De Tareas 214
Plataformas para el desarrollo de Agentes Móviles 125
Ara 127
D'Agents 126
Java....... 127
JVM 128
Programación Multihilo 129
RMI y Serialización....... 129
Sandbox....... 129
Telescript 125
Portabilidad y Seguridad 123
Propiedades de agentes 95
Protección del Estado de Datos

Mediante Composición de Matrices 301
Mediante Duplicación de Agentes Móviles 297
Mediante Funciones Homomórficas 301
Mediante Navegación en Redes Seguras .. 295
Mediante PRAC's 298
Mediante Técnicas de Duplicación de Datos 296
protocolo 427

S

SAHARA
Arquitectura de Seguridad Integral ... 241, 255
Asignación de Permisos 276
Autenticación de Agentes 255
Autorización de Privilegios 263
Cumplimiento de Privilegios 279
Esquema Global de Interacción 245
Lugares 248
Panorama General de Funcionamiento 251
Política de Seguridad 270
Protección de Agentes
De sus Datos 294
Del Código 294
Protección en la Transmisión
SSL 288
Regiones 249
Tipos de Autoridades 247
Tipos de Permisos 266
Seguridad en Sistemas de Agentes 241
Conceptos Básicos 244
Servicios de CORBA 37
El servicio de actualización de objetos 55
El servicio de ciclo de vida de objetos 39
El servicio de colección 50
El servicio de consultas 48
El servicio de control de concurrencia 43
El servicio de eventos 40
El servicio de exteriorización 52
El servicio de licencias 53
El servicio de nombrado de objetos 37
El servicio de persistencia de objetos 44
El servicio de propiedad de objetos 54
El servicio de relación 50
El servicio de seguridad de objetos 54
El servicio de sincronización de objetos 54
El servicio de transacción de objetos 41
El servicio negociación 38
Sistemas gestores de bases de datos orientados a objetos 46
SIG 120
Supercomponentes 24

T

Tecnología de agentes móviles 108
El lenguaje 108
La máquina o el interprete 109
Protocolos 109
Tendencias de los sistemas de agentes móviles actuales 207
Trabajo Relacionado con la Investigación de SAHARA 375
Aproximación de Fritz Hohl para Resolver el Problema del Servidor Malicioso 383
El Modelo de Seguridad de los Aglets 375

V

Ventajas de los agentes móviles 113