Sistema de Métricas para Tiempo Real en Aplicaciones Java

Aquilino A. Juan Fuente
Profesor de la Universidad de Oviedo
aquilino@lsi.uniovi.es
Raul Izquierdo Castanedo
Profesor de la Universidad de Oviedo
ric@pinon.ccu.uniovi.es
Juan Manuel Cueva Lovelle
Catedrático de Escuela de la Universidad de Oviedo
cueva@pinon.ccu.uniovi.es
Benjamín López Pérez
Titular de Escuela de la Universidad de Oviedo
benja@pinon.ccu.uniovi.es
Luis Joyanes Aguilar
Catedrático de la Universidad Pontifica de Salamanca

Resumen

Los sistemas de métricas de software tradicionales se han centrado fundamentalmente en las métricas de procesos, de productos y de recursos [FEN91]. Los principales objetivos de todos los sistemas de métricas desarrollados hasta el momento son el control y la estimación de los proyectos, la calidad del desarrollo o los costes y esfuerzos estimados durante la fase de mantenimiento de la aplicación.

En algunas ocasiones se han utilizado las métricas en tiempo de ejecución, normalmente como estimadores de velocidad, testing de ejecución, profiling o para estimación del coste de servicios, pero en ningún caso con intención de control y monitorización de la aplicación. Con estos cometidos se han utilizado desleídas en el código y despojadas de intencionalidad.

En este artículo se trata de dar un nuevo enfoque a las métricas. Lo primero es definir qué se entiende por métrica en tiempo de ejecución, después se separará lo que, en un diseño, atañe a proceso algorítmico puro y a métrica de atributos insertos en la aplicación durante una ejecución de ésta.

Por último, se definirá un framework básico para definición de Métricas en Tiempo de Ejecución (MTE) en programas realizados en lenguaje Java.

Abstract

Traditional software metrics systems has been pointed mainly on process, products and resource ones [FEN91]. Main objectives in all the metrics systems developed until today was

control and projects estimation, development quality and costs and estimated efforts in the application maintainement phase.

Occasionally any kind of run-time metrics was used for speed measurement, execution testing, profiling and services costs. But it was never used for monitoring and control applications. For this purposes, they was used in an unclear way inside code.

This article try to give to metrics a new view point. First we define what run-time metrics is, then we will divide what is algorithmical and what is metrical code.

Lastly, we will define a basic framework for defining run-time metrics (RTM or MTE) in programs in Java languaje.

1. Introducción

La informática ha evolucionado desde un enfoque puramente artesanal hasta llegar a uno más científico, donde cada vez es mayor la necesidad de justificar de una manera razonada tanto los diseños como la propia programación. Es, por tanto, cada vez más importante disponer de una metodología y de un sistema de métricas adecuados a nuestras necesidades.

Las ciencias y las ingenierías, para hacer mediciones, utilizan teorías formales generadas a partir de las observaciones empíricas y las expresan mediante notación matemática. La informática, por tanto, como ingeniería, debe utilizar también esas teorías formales.

Por todo ello y por la importancia que concedemos al hecho de medir, no podemos utilizar cualquier sistema de medida, sino que éste debe estar sujeto a una teoría: LA TEORÍA DE LA MEDICIÓN.

La ingeniería informática está aún en un estado de unificación de las observaciones empíricas y las teorías formales. Este estado, que es debido principalmente a la enorme complejidad del software, impide la penetración de los avances teóricos en la industria, reduce la calidad de los desarrollos de software y de los sistemas relacionados con el software, y crea desunión dentro de la profesión.

El objetivo de una teoría de la medida es el de lograr que la descripción que las métricas nos aportan del sistema sea objetiva, exacta, reproducible, segura y significativa.

Pero además la informática es una ciencia que trata de modelar el conocimiento humano, por lo que es importante que también modele las propias métricas, al fin y al cabo producto de la observación humana y del conocimiento formal.

Muchas son las cosas que se pueden medir en un proyecto de software y muchos autores han tratado sobre este tema. La mayor parte de las métricas (o pseudométricas) definidas están orientadas a los procesos de desarrollo o a los productos ya elaborados, pero casi siempre desde un punto de vista de vista estático del sistema y no desde el punto de vista de que una aplicación es un proceso que evoluciona y esa evolución también debe ser medida.

Este artículo tratará sobre la medición de la evolución de los sistemas. A este tipo de métricas las llamaremos métricas en tiempo de ejecución.

2. Objetivo

El objetivo de este artículo es definir el concepto de Métrica en Tiempo de Ejecución (MTE ó Run-time Metric RTM) y definir un posible framework de soporte para aplicaciones en Java.

3. Descripción de Métrica en Tiempo de Ejecución

Podemos definir una métrica en tiempo de ejecución como el resultado formalmente expresado de la observación del comportamiento de un atributo (simple o complejo) que forma parte de la ejecución o de la evolución de una aplicación.

Desde este punto de vista, el sistema que mida debe estar en ejecución al mismo tiempo que el sistema observado y medirá atributos internos del sistema observado.

Por transitividad, estos atributos podrían corresponder a medidas del mundo real capturadas por el sistema.

El problema del planteamiento de una métrica de estas características es que se necesitan una serie de premisas para que, tanto la medición como la monitorización se separen de la ejecución normal del sistema observado, tanto en su concepción como en su ejecución.

4. Estado Actual del Arte

Los sistemas de métricas para tiempo de ejecución más comunes hoy en día son los usados en los profilers o aplicaciones para probar las aplicaciones. Este tipo de aplicaciones usan sistemas de métricas en tiempo de ejecución para medir tiempos, buscar cuellos de botella en las aplicaciones, medir capacidades máximas, etc.

Si bien este tipo de aplicaciones son de un gran interés y están dentro del concepto definido como MTE, tienen una limitación importante y es que solo sirven de meros observadores y no definen las acciones a tomar en caso de que el sistema evolucione de manera desacorde con lo deseado.

Además otro inconveniente es que al estar absolutamente fuera del sistema medido, no pueden ser aprovechados para ser parte de él, definiendo parte de su lógica interna y por tanto tomando parte activa en la evolución del sistema, si así fuera requerido.

Haciendo un símil electrónico, los profilers son aparatos de medida como lo es un osciloscopio o un voltímetro y el sistema definido en este artículo es un elemento activo del sistema como pueda ser un circuito recortador de onda o un estabilizador de tensión.

El objetivo de este artículo no es definir nuevas métricas, sino dar el soporte para la instalación de métricas en tiempo de ejecución definidas ad-hoc para cualquier sistema y para la definición de ciertos patrones de comportamiento inherentes a los sistemas de software.

5. Separación entre algoritmo y Métrica

Definir toda una metodología de trabajo para el diseño de los sistemas de métricas en tiempo de ejecución es algo que está en proceso de desarrollo dentro del OOTLAB en la

Universidad de Oviedo. Pero en este artículo podemos definir unos breves apuntes de las condiciones ideales que deberían tener los elementos implicados en dicha metodología.

En concreto, los lenguajes de programación, deberían tener un soporte específico de métricas y no, como en la actualidad, en que la métricas deben ser definidas como añadidos e implementadas sobre el propio código desvirtuando en gran manera la pureza de los algoritmos y provocando aún más efectos laterales en el código.

A modo de ejemplo, se podría citar el propio sistema que control de excepciones de los lenguajes orientados a objetos actuales. En concreto el sistema usado por C++ y Java, se pueden considerar un primer paso en este objetivo de separar lo que es la propia lógica de la aplicación por un lado y las operaciones de control y recuperación de errores y excepciones por otro. Otro ejemplo lo tenemos en Eiffel con la definición de la Programación por Contrato [MEYER97].

Del mismo modo, debería haber también un sistema específico de inserción de métricas en el código.

Este sistema debería cumplir con las características de tener una separación clara entre el código destinado a la lógica de la aplicación y el destinado a la medición, a la monitorización o al control.

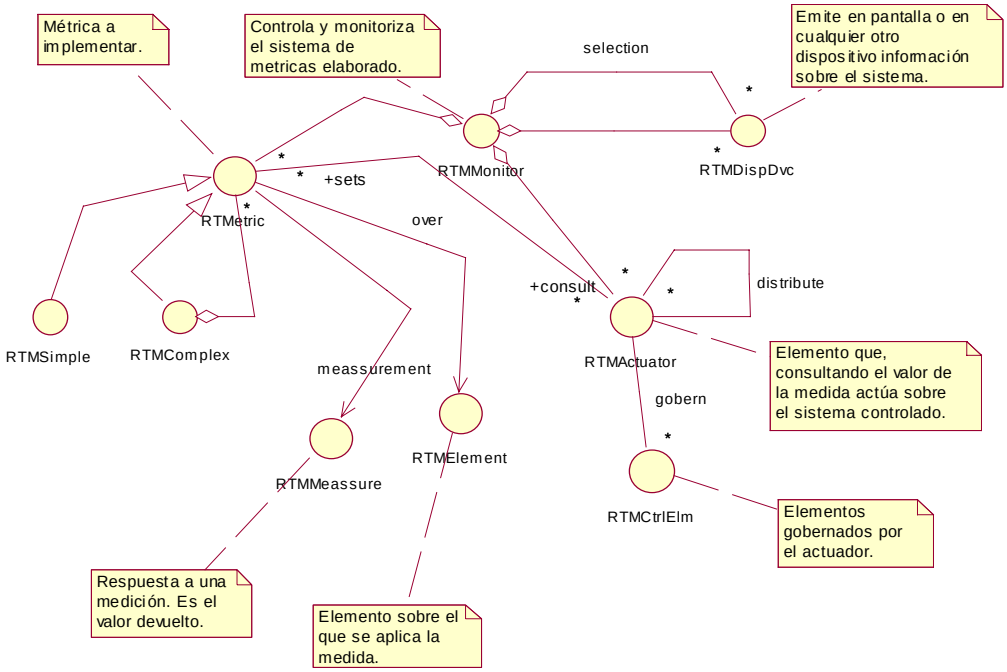
El objetivo de definir ampliaciones en los lenguajes para incluir ciertas operaciones adicionales a la lógica de la aplicación, no está cerrado en absoluto, pero tampoco puede ser planteado de manera frívola, ya que en este caso podríamos llegar a múltiples sublenguajes dentro de un lenguaje inicial de partida y se debería limitar este tipo de crecimiento descontrolado. En opinión de los autores de este artículo, sólo merecen esta distinción especial por el momento el sistema control de excepciones, la programación por contrato [MEY98] y el sistema de métricas.

Sobre la programación por contrato, existen varios resultados de inclusión de programación por contrato en Java [AQU98-1 y AQU98-2] que demuestran la posibilidad de separar claramente la parte de implementación pura del algoritmo de los controles de precondiciones, postcondiciones, asertos e invariantes.

Así pues, el objetivo final sería la inclusión de un sistema de métricas dentro del código siguiendo estos principios de transparencia. Pero este planteamiento conlleva, previamente, el diseño de un framework que soporte las métricas sobre el estándar actual de cada lenguaje orientado a objetos. El objetivo de este artículo es la definición de este framework básico para poder plantear un sistema de mediciones en aplicaciones implementadas en Java. En otros artículos en preparación se definirán las características del lenguaje.

6. Framework Básico para Métricas en Tiempo de Ejecución en Java

En la elaboración del framework se ha seguido el esquema de diseñar una serie de interfaces que implementan el diseño básico del sistema y que se desarrollarán siguiendo modelos de patrones de diseño [GAMM94]. Este esquema se puede ver en el siguiente diagrama.



El esquema consta de siete interfaces básicas que colaboran dentro del sistema. De ellas, las más importantes son `RTMetric`, `RTMMonitor` y `RTMActuator`.

El monitor (`RTMMonitor`) debe ser el centro de control del sistema y de él depende el funcionamiento del resto. Básicamente debe identificar las métricas a usar (`RTMetric`) y los actuadores (`RTMActuator`) y ponerlos en comunicación para que colaboren en los objetivos planteados al sistema de monitorización.

6.1. Colaboraciones

6.1.1. RTMMonitor

Es la interfaz principal del sistema y es el centro de control para cualquier que utilice el framework. Registra todas la métricas y los actuadores y monitoriza para el usuario aquellas medidas que se hayan definido. Debe estar en un hilo de ejecución independiente.

6.1.2. RTMetric

Es la interfaz de la métrica que se haya implementado. Puede ser simple o compleja y para implementar esta última se utiliza el patrón *composite* (`RTMSimple` y `RTMComplex`).

6.1.3. RTMMeassure

Es la interfaz de la medida realizada. Cada vez que se haga una medida se devolvera un objeto que cumpla esta interfaz para que pueda ser interrogado por el monitor.

6.1.4. RTMElement

Cada objeto que contenga atributos o características que vayan a ser medidas deberá implementar esta interfaz para que `RTMMeassure` pueda interrogarlo.

6.1.5. RTMActuator

Implementa los actuadores que interaccionan con el sistema. A nivel de interfaz no está definido un patrón concreto para realizar este subsistema, pero en general puede ser un patrón *composite* para tener a todos los actuadores que colaboran con una `RTMetric` y un patrón *chain of responsibility* para pasar los eventos. La relación entre `RTMetric` y `RTMActuator` puede ser la de un *obserer* de éste último para actuar en los cambios de valor de `RTMetric`.

Otros patrones que pueden encajar son *strategy*, *state*, *command*, etc.

6.1.6. RTMCtrlElm

Esta interfaz debe ser implementada por el elemento bajo control del actuador, a través de ella, el `RTMActuator` puede controlar el comportamiento de este último. El patrón aquí puede ser de nuevo un *observer*.

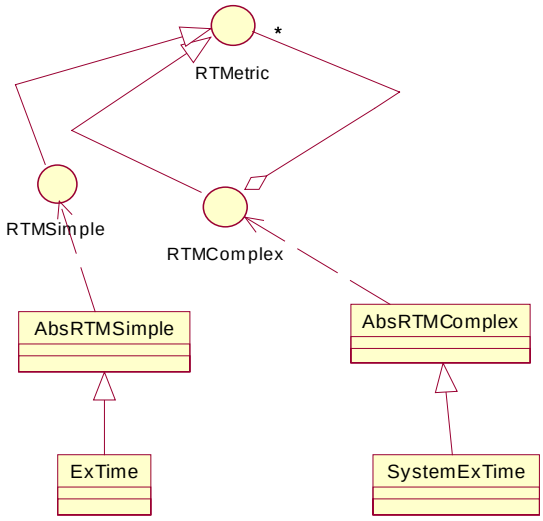
6.1.7. RTMDispDvc

Implementa la interfaz para componer la información que se mostrará al usuario. Aquí pueden encajar varios patrones, la relación entre esta interfaz y `RTMMonitor` puede implementarse a través de un *observer*, asimismo, las relaciones entre los diferentes objetos de esta interfaz encajan varios patrones (*composite*, *chain of reponsibility*, etc.)

6.2. Implementación de la solución

Entre el nivel de interfaces básicas del framework y el nivel de clases concretas capaces de ser instanciadas en objetos, se debería pasar por un nivel intermedio de clases abstractas que definan de un modo más preciso la solución para un conjunto importante de diferentes sistemas e implementen algunos patrones de diseño que acerquen la solución concreta.

A modo de ejemplo, `RTMetric` puede ser como se ve en siguiente diagrama.



7. Conclusiones

Como se ha visto en el artículo, las métricas en tiempo de ejecución son una parte importante de las métricas de software.

Mediante estas métricas es posible plantear sistema de monitorización y separar la ejecución de algoritmos propios del sistema a solucionar de los controles específicos que se necesita para dimensionar adecuadamente el sistema en ejecución.

El framework nos permite definir a un nivel de abstracción importante cualquier métrica que se desee implementar, así como cualquier tipo de actuación sobre el sistema en función de los valores leídos por las métricas.

8. Bibliografía

[AQUI98-1] ESPECIFICACIÓN SEMÁNTICA DE COMPONENTES EN JAVA, PROGRAMACIÓN BAJO CONTRATO (HAN-02).
Aquilino A. Juan, Raul Izquierdo y Juan M. Cueva.
Universidad de Oviedo. 1998.

[MEYER97] OBJECT ORIENTED SOFTWARE CONSTRUCTION.
Bertrand Meyer.
Editorial Prentice Hall. 1997, ISBN: 0-13-629155-4

[AQUI98-2] ESPECIFICACIÓN FORMAL DE COMPONENTES EN JAVA (PROGRAMACIÓN POR CONTRATO)
Aquilino A. Juan, Raul Izquierdo, Juan M. Cueva y Carmen Viejo Vigil.
Universidad de Oviedo. 1998. - JJOO –1998, Proceedings, pág 131-138.

[GAMM94] DESIGN PATTERNS
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Editorial Addison Wesley. 1994. ISBN: 0-201-63361-2.

[FEN96] SOFTWARE METRICS: A RIGUROUS & PRACTICAL APPROACH
Norman E. Fenton, Shari Lawrence Pfleeger
Thonson Computer Press, ISBN: 1-85032-275-9 (1996)

[FEN91] SOFTWARE METRICS: A RIGUROUS APPROACH
Norman E. Fenton
Thonson Computer Press, ISBN: 1-85032-242-2 (1991)

9. Páginas Web

http://www.ootlab.uniovi.es	Página web del Laboratorio de Tecnologías de Objetos de la Universidad de Oviedo.
http://www.di.uniovi.es/~aquilino	Página web de Aquilino A. Juan.