A Generic e-Learning Multiparadigm Programming Language System: IDEFIX Project *

J. E. Labra Gayo
labra@lsi.uniovi.es

J. M. Morales Gil
jmmoral@correo.uniovi.es

A. M. Fernández Álvarez
alb@correo.uniovi.es

H. Sagastegui Chigne
hsagastegui@hotmail.com

Department of Computer Science
University of Oviedo
C/ Calvo Sotelo S/N, 3307, Oviedo, Spain

ABSTRACT

We describe the main features of an Internet based distance learning environment that we are developing to teach a course on logic and functional programming for computer science students. The main goal of the system is to provide a minimal set of capabilities for a generic interpreter that will be instantiated for different programming languages and paradigms. The architecture of the system is based on the use of XML and web services to facilitate the integration and independence between the different development platforms and systems.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors—*interpreters, run-time environments*

General Terms

Design Languages

Keywords

Functional Programming, Logic Programming, Web Services, XML, Interpreters

*Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE'03, February 19-23, 2003, Reno, Nevada, USA.
Copyright 2003 ACM 1-58113-648-X/03/0002...$5.00.

1. INTRODUCTION

Our University takes part in the development of a *Virtual Campus* to share e-learning courses with six other spanish universities. In that system, the AulaNet project [3] from our University has incorporated the course *Logic and Functional Programming* in 2001/2002, which is taught by the first author of this paper. In the course, we present the functional and logic paradigms with their corresponding programming assignments in Haskell and Prolog languages.

There are several tools in the market to publish distance learning courses [10]. In the case of the AulaNet project a specialized tool was implemented using Java and Servlets. That tool had some appealing features but the teaching resources were not compatible with other courses offered in the *Virtual Campus*. To solve that problem, the WebCT [4] tool was adopted last year. The tool allows the designer of a course to incorporate different teaching resources mainly based on HTML or PDF format. Although the tool has a nice user interface, we have considered that a more flexible approach would allow us to adapt our teaching resources to other systems. To that end, we have developed our own XML based vocabularies for lecture notes and assignments. Nevertheless, given the particular intricacies of teaching programming languages, we have also considered developing an Internet based development environment that would allow students to write, test and submit their programming assignments from different machines connected to a server. At the same time, it would also allow the system to monitor students progress. The project is called IDEFIX (Integrated Development Environment based on Internet and eXtensible technologies) project [5] and is based on a decentralized architecture based on XML and Web Services. In the rest of the paper, we give an overview of these technologies, their application to our system, and the architecture of the IDEFIX project.

2. XML AND WEB SERVICES

XML [7] is rapidly emerging as the widely adopted standard for Internet information exchange and representation. It allows a common syntax for specialized vocabularies, which can syntactically be validated. While HTML has a fixed set

of tags with a predefined semantics, XML allows the application developer to define specialized vocabularies. XML documents can be processed with a variety of tools. In particular, XSLT allows quick transformations based on the document tree.

In our case, we have defined two vocabularies: one for lecture notes and the other for programming assignments and we have made mappings of those vocabularies to HTML and PDF. A nice feature of this approach is that the XML document can also be adapted to extract the source code and to load and execute it. In that way, all the definitions can be tested and, in the case or programming assignments, we can develop tools to automatically check the student programs results.

As an example, in figure 1, we present an XML fragment of a programming assignment in our XML vocabulary. Notice the use of CDATA sections to include Haskell code using the literate programming convention. In this way, the same XML file can be compiled using standard Haskell implementations. The XML file also includes some tests which can be visible or hidden. Visible tests are given to the students while hidden tests are used to automatically check students programs. Figure 2 presents the PDF output file for students generated from the XML file.

Web services can be considered an umbrella term to describe components that are addressable and available using Web technology. They use standard communication protocols built on XML. Specifically, the current generation web services use SOAP for message format, WSDL for service description and UDDI for service discovery. There is a growing interest in the software industry to adopt this architectural style [2].

One of the advantages of web services is that they facilitate cross-platform development in a lightweight communication protocol. In our case, we divided the system into different components which are being independently developed by students.

```
<assignment name="mean">
<desc>Write a function <code>mean</code>
  that calculates the mean of a list of
  ints</desc>
<type><![CDATA[

 > mean :: [Float] -> Float

]]>
</type>

<test type="public">
<query>mean [10,20,30]</query>
<answer>20</answer>
</test>

<test type="hidden">
<query>mean [1]</query>
<answer>1</answer>
</test>

<test type="hidden">
<query>mean [1,-1]</query>
<answer>0</answer>
</test>
<solution>
<![CDATA[

> mean xs =
>  sum xs / fromIntegral (length xs)

]]>
</solution>
</assignment>
```

Figure 1: Example of Programming assignment in XML

3. REQUIREMENTS OF THE IDEFIX SYSTEM

The main requirements of the IDEFIX system are:

- *Programming Language Independence*: The adaptability of the system to teach different programming languages and paradigms is an important challenge in a course like *Logic and functional programming*. Currently, we teach *Prolog* and *Haskell* but we are considering to use *Curry* in a near future. Moreover, we would like the system to be useful for other programming language courses with object-oriented and imperative languages like *Java* or *C*. This requirement implies a flexible design that allows the course designer to change the configuration of tools used by the system, like the set of programming assignments, the interpreters/compilers and their interfaces with the different tools that control the transmission of information through Internet.
- *Automatic Evaluation of Student Assignments*: The number of students that take distance-learning courses is growing each year. In the case of programming assignments, it is a difficult task to evaluate the student work without a direct contact with the student. Although there are some tools automatically assess student's works [11, 17, 14, 18], we have considered developing our own tool using the XML framework. The programming assignment designer can write a set of visible tests as well as a set of hidden tests. Once the student writes a solution, the automatic evaluation tool runs the program and checks that all the tests are achieved. Another possibility that we would like to consider is to specify program properties that could be checked using tools like QuickCheck [9].
- *Encourage Collaborative working environments*: Although sometimes distance learning can favour individual work, the tools that it gives like email, chats, discussion groups, etc. offer the possibility to develop collaborative working environments [13] were the student is not just a passive element, but can participate in the development of the system.

  In the case of programming languages, the number of error messages of each system and the infinite possi-

ASSIGNMENT 1. *Write a function* **mean** *that calculates the mean of a list of floats*

```
mean :: [Float] -> Float

? mean [10,20,30]
20
```

Figure 2: Output file for students in PDF

bilities that appear when one writes a program make it necessary the development of some kind of collaborative programmer assistant system. It is no surprise the profusion of Wiki sites devoted to different programming languages where the users can add information in an interactive way. In this project we intend to develop similar systems where the students can participate adding information and solving other students problems.

- *Adaptability of user interface*: An important point in the development of web sites is the user interface. In our research group we have done some work on systems that remotely check web navigability [12]. These tools could be incorporated to provide adaptable user interfaces that improve system usability.

- *Internationalization*: Although the system is mainly being developed for Spanish speaking users, we consider a very important feature of these kind of tools that they can be easily translated to different languages. We use English as the implementation language and even for the communication between developers. The user interface can be customized to different languages.

- *Compatibility with other systems*: Currently, our system must interoperate with WebCT and the AulaNet tools. However, we want it to survive to the future introduction of new tools. To that end, an important design decision is to select standard technologies or to provide mappings between our formats and other formats. The use of XML facilitates the translation of teaching material to different formats. As we have mentioned, we have already made translations to HTML and PDF. We are also studying possible translations to other distance learning standard specifications [6, 15, 16].

4. ARCHITECTURE OF THE IDEFIX SYSTEM

The architecture of the IDEFIX system is presented in figure 3. The key feature of this architecture is that most of the components are connected through Internet using Web Service's protocols. In this way, the system offers an open architecture that will allow users to add more components and features in the future. The current components are:

- *Generic Interpreter*: This component offers a minimal set of capabilities that all the interpreters/compilers should accomplish like initialize, load a program, run

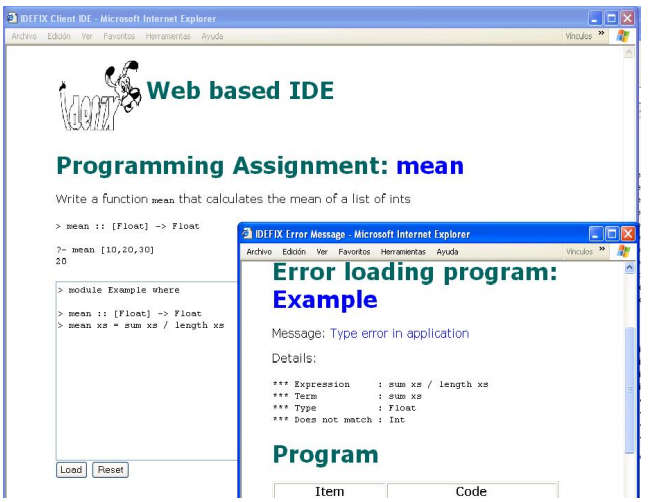

Figure 4: Client IDE

a query, etc. This is the main web service and defines programs as a list of items. Each item has a specific role, which depends on the programming language and can be used for debugging information. With this representation, we can offer the client a minimal set of information about the program without compromising programming language independence.

- *Client IDE*: This component controls the user interface and acts as a client of the Generic Interpreter web service. We expect to develop several clients with different features. In fact, at this moment we have developed a simple text-based menu interface in C#, we are developing a more user-friendly web-based interface in ASP.NET and we expect more clients to be built using other technologies. Figure 4 represents a simple client IDE running the previous example program.

- *Haskell wrapper*: We have developed a simple wrapper that translates Haskell programs to/from a list of items. Currently, we are using Hugs [1] as the Haskell interpreter. The reason is that Hugs implementation offers a server component that can be encapsulated in a DLL, which facilitates the link with the .NET platform. In the future, we would consider other Haskell implementations, like the GHC compiler. The XML vocabulary for error messages is independent of the programming language. The different wrappers can capture some information like the line number which can help the client IDE to visualize this information in a user friendly way.

- *Prolog wrapper*: We are developing a Prolog wrapper around our own Prolog system. Our system has been implemented in Java and offers a full Standard Prolog implementation. However, we will also provide wrappers to other Prolog systems, like SWI-Prolog or Ciao-Prolog.

- *Assignment Manager*: This component keeps track of programming assignments. It maintains its own

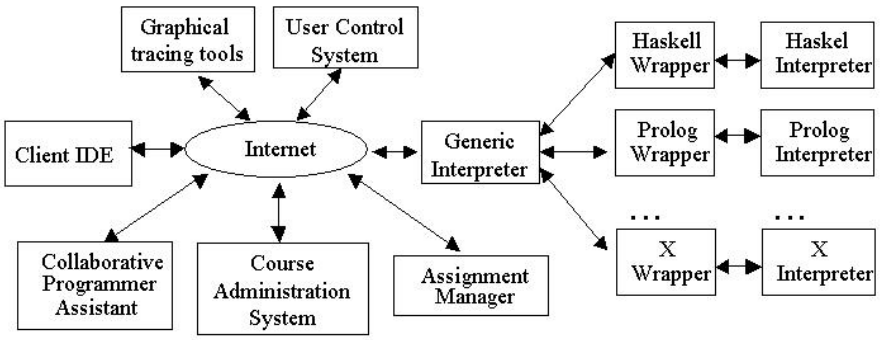

Figure 3: Architecture of the IDEFIX System

database of programming assignments allowing the students to select and solve the different ones.

- *Course Administration System*: This component is responsible of administrative tasks like student records, grades, etc. It must also be able to communicate with other administrative legacy software from the university.
- *Collaborative Programmer Assistant*: As we have said, in our course we want to encourage collaborative work. To that end, this component will provide a generic helping system in the spirit of the popular Wiki sites. The only difference will be that our system will be based on web services and will offer a generic functionality adapted to our framework.
- *Graphical tracing tools*: We have developed an Internet based Logic programming tracer tool. This is an independent system based on our own Prolog Interpreter. The interpreter has been written in Java and the web based user interface has been developed in JSP and VML. Figure 5 presents a screen capture of the running system. The system draws the SLD resolution tree by user demand.
- *User control system*: This component will be in charge of user control. An important feature is that it will not be embedded in the application but will be accesible through web services. This component is similar to Microsoft's Passport. In fact, we would like to adopt a standard and public web service for user control.

5. CONCLUSIONS

At this moment, we have already developed some working prototypes, which confirm that this kind of decentralized systems can be implemented. Most of the implementation work is being done by students as part of their individual final year projects. We have considered more important to establish a well-defined common interface than to force the students to adopt a particular development platform. In this way, the system is formed by a combination of different implementation languages (Java, C#, Haskell, Prolog, etc.) and platforms (mainly Windows and Linux). The glue between this variety of systems are XML and web services. We can give some assessment based on our experience with these technologies:

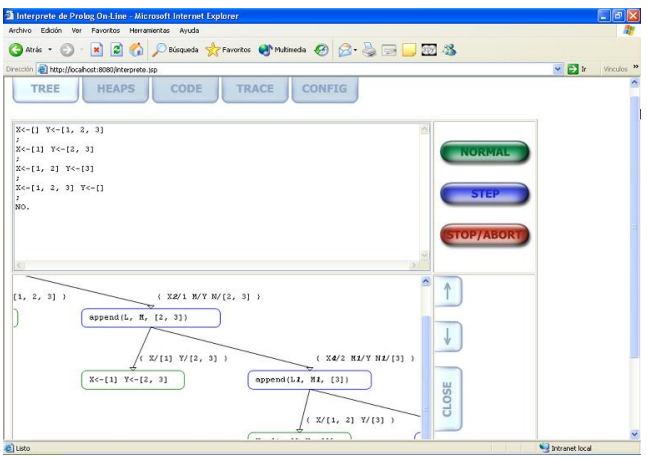

Figure 5: Screen capture of the Logic Programming tracer

- *Emphasis on interfaces*: Web service technology emphasizes the definition of standardized interfaces. Applications can obtain new functionality from third party components and the lightweight communication protocols allow a very decentralized architecture. This enables the development of software products integrating different platforms and systems. In our case, our research group web server is running Apache on Linux while the AulaNet software is based on the Microsoft platform. At the same time, some students prefer to develop under Windows while others are free software followers. Using web services, these different points of view can live together.
- *Industrial adoption*: It is expected that XML and web services will be adopted by the main software industries. At least, they have participated in their design and there are a good number of development libraries for the different platforms. At the same time, however, this technology is still in its infancy and there are some incompatibilities between implementations: sometimes because they do not follow the standard and sometimes because the standard changes.
- *Performance*: Although at this moment we have a

small number of students, the performance of the system is an important issue. This technology adds a new indirection level using XML as the base format which is not specially oriented towards efficiency. In fact, XML files can be 3 to 20 times as large as a comparable binary or alternate text file representation. Using SOAP, the marshalling that occurs between XML messages and application data can also cause system performance to be quite poor.

- *Security*: In the case of distance learning courses, specially when trying to assess student's work, the service's security model is crucial. Although there are some standards for encoding security credentials in SOAP messages, they are not yet widely accepted. Another issue related with security is the quality of service. In the hypothetical case that someone was paying money for web service consumption, what would happen if a chain of web services could not solve a query? Who is responsible if something goes wrong? At this moment, we have not found good answers for these questions, although we expect more research in this direction.

- *Evolution of services*: As any software product, web services could change their interfaces and workflows between different versions. What happens to the clients when interfaces change? We are not aware of any systematic solution to solve the problem of upgrading services and how that affects existing client software. In this field, it would be interesting to adapt some work already done in the development of component-based software [8].

6. REFERENCES

[1] Home page of the Hugs system. `http://www.haskell.org/hugs`.

[2] World wide web consortium workshop on web services. `http://www.w3.org/2001/03/WSWS-popa/paper51`, Apr. 2001.

[3] AulaNet project homepage. `http://www.aulanet.uniovi.es/`, 2002.

[4] Home page of the WebCT system. `http://www.webct.com`, 2002.

[5] Integrated Development Environment Frameworks based on Internet and eXtensible Technologies: IDEFIX project. `http://www.di.uniovi.es/aplt/idefix`, 2002.

[6] Advanced Distributed Learning. Sharable content object reference model. `http://www.adlnet.org`.

[7] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (1.0). `http://www.w3.org/TR/REC-xml`, Oct. 2000. 2nd. Edition.

[8] A. Cernuda, J. E. Labra, and J. M. Cueva. Itacio: A component model for verifying software at construction time. In *Third International Workshop on Component Based Software Engineering, 22nd. International Conference on Software Engineering*, Limerick, Ireland, June 2000.

[9] K. Claessen and J. Hughes. QuickCheck: A lightweight tool for random testing of haskell programs. In *International Conference on Functional Programming*, pages 268–279. ACM-SIGPLAN, 2000.

[10] T. Clear, A. Haataja, J. Meyer, J. Suhonen, and S. A. Varden. Dimensions of distance learning for computer science education. *SIGCSE Bulletin*, 33(2), 2001. ITiCSE 2000 Working Group Reports.

[11] K. M. Dawson-Howe. Automatic submission of programming assignments. *SIGCSE Bulletin*, 27(4):51–53, 1995.

[12] B. M. González, J. E. Labra, and J. M. Cueva. Web navigability testing with remote agents. In *Second ICSE Workshop on Web Engineering, 22nd. International Conference on Software Engineering*, page 311, Limerick, Ireland, June 2000.

[13] M. Guzdial. Use of collaborative multimedia in computer science classes. *SIGCSE Bulletin*, 33(3):17–21, Sep. 2001. 6th Annual Conference on Innovation and Technology in Computer Science Education.

[14] D. M. Huizinga. Identifying topics for instructional improvement through on-line tracking of programming assignments. *SIGCSE Bulletin*, 33(3):129–132, Sep. 2001. 6th. Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education.

[15] IEEE. Learning technology standards comittee. `http://ltsc.ieee.org`.

[16] IMS Global Learning Consortium. `http://www.imsproject.org`.

[17] C. G. Rodríguez, L. L. Díaz, P. P. Costanza, C. P. Flores, R. M. Unanue, and J. A. V. Iturbide. EXercita: Automatic web publishing of programming exercises. *SIGCSE Bulletin*, 33(3):161–164, 2001.

[18] R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. *SIGCSE Bulletin*, 33(3):133–136, Sep. 2001. 6th. Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education.