Using Domain Specific Languages to Instantiate OO Frameworks

Christiano O. Braga, Marcus Felipe M. C. da Fontoura, Carlos José P. de Lucena,
and Leonardo M. de Moura
Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil
e-mail: [cbraga, mafe, lucena, moura]@inf.puc-rio.br
PUC-RioInf.MCC28/98 Julho, 1998

ABSTRACT

Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks. An object-oriented framework captures the common aspects of a family of applications. The framework also captures the design experience in producing the application, and thus, allows the designers and implementers to reuse this experience at the design and code levels. Despite of being a powerful design solution, frameworks are not always easy to use. This paper describes a technique that captures domain concepts in domain specific languages (DSL) to help the framework user to create a specific application in an easier way, without worrying about implementation decisions and remaining focused on her problem domain. The specification written in the DSL is then translated to the framework instantiation code using a transformational system.

KEY WORDS: object-oriented frameworks, domain-specific languages, instantiation process, transformational systems.

RESUMO

Pesquisas têm mostrado que altos níves de reutilização de software podem ser alcançados através do uso de frameworks orientados a objetos. Um framework orientado a objetos possui funcionalidades comuns a famílias de aplicações. O framework também captura a experiência de design na construção de aplicações, permitindo que designers e implementadores reutilizem essa experiência nas etapas de design e codificação. Apesar de serem uma poderosa solução de design, frameworks não são sempre fáceis de usar. Este artigo descreve uma técnica que captura os conceitos do domínio em linguagens de domínio (DSL), ajudando o usuário do framework a criar aplicações específicas de maneira mais simples, sem se preocupar com decisões de implementação e permanecendo focado no domínio do problema. A especificação escrita na DSL é então transformada no código de instanciação do framework através de sistemas transformacionais.

PALAVRAS CHAVES: frameworks orientados a objetos, linguagens de domínio, processo de instanciação, sistemas transformacionais.

1. Introduction

There are various application areas that are not established yet and for which new ideas and models are presently under development and evaluation. These are domains in which the possibility of rapidly building new applications is essential and strategic from a practical point of view. Examples of application domains that can be classified in this category include Web-based Education, electronic commerce, biology, and financial market.

An interesting strategy for developing new applications for these domains is the development of object-oriented application frameworks [1, 2]. Frameworks can reduce the costs of developing applications since it allows designers and implementers to reuse their previous experience with problem solving at the design and code levels. Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks. An object-oriented framework captures the common aspects of a family of applications, which can be seen as a domain theory [3].

Although object-oriented software development has experienced the benefits of using frameworks, a thorough understanding of how to identify, design, implement, and change them to meet evolving requirements needs are still object of research [4]. Techniques such as design patterns [1, 5], meta-object protocol [6], refactoring [7], and class reorganization [8] have been proposed to support framework development and cope with some of the challenges.

Therefore, framework development is very expensive not only because of the intrinsic difficulty related to capturing the domain theory but also because of the lack of appropriate methods and techniques to support framework design and instantiation.

Frameworks like Microsoft MFC [9], ET++ [10], Vix [11] are interesting references. All examples have one thing in common: they are frameworks for the application interface domain. The reason why it is difficult to think about other different examples is because they are not easy to find in the literature due to the difficulties of developing and using frameworks.

The most common way to instantiate a framework is to inherit from some abstract classes defined in the framework hierarchy and write the code that is called by the framework itself. However, it is not always easy to identify which code and where it should be written since frameworks class hierarchies can be very complex, especially for non-expert users.

Therefore, the "common" instantiation process is not always the ideal way to describe a particular application. This happens because of several facts:

- the language used to build and use the framework is a wide spectrum language, where it is not always easy to express the user intentions [12]. A short example would be the definition of event handlers in Microsoft MFC. The user has to write a macro to register her methods that will be called when a button is pressed or a list is scrolled. With an interface definition language [13], this task becomes easier and more intuitive;
- the complexity of framework class hierarchies and the difficulty of finding the points where the code should be written, that are the framework hot-spots or flexible points [5, 14].

This paper presents a different process for framework instantiation, where a particular application is described by a domain specific language (DSL) [15, 16, 17, 18] that is designed precisely for that application. The use of the DSL allows the designer to develop quickly and effectively a complete software system [17]. In this way, the technique basic idea is to capture domain concepts in a DSL, which will help the framework user to build code in an easier and specific way, without worrying about implementation decisions and remaining focused on her problem domain. The specification written in the DSL is then translated to the framework instantiation code using a transformational system [19, 20].

2. Domain Specific Languages

A domain specific language is used to formally specify software designs [15]. It is a formal language that is expressive over the abstractions of an application domain. Common examples of DSLs are:

- Tex and Latex, text formatting languages;
- YACC, a parser generator;
- Mathematica, an extensible language for mathematical modeling;
- Schema description and query languages for databases.

An advantage of using a DSL is that the domain expert can express domain specific concepts directly, rather than encoding them. This allows the domain expert to formalize the specification of a software solution immediately instead of communicating the specification informally to a software specialist who may be less familiar with the intended application. When designing a DSL two important criteria should be kept in mind:

- The DSL must be intelligible to a domain expert;
- The semantics must allow a specification expressed in a DSL to be translated into effective procedures that realize the specification.

Typically, such specification will provide static and dynamic constrains on an artifact of the application domain, or will specify its dynamic behavior. Often, a syntax driven editor could be used to help an application designer to formulate an application in the DSL.

3. The Instantiation Process

This section shows how DSLs can be used to instantiate frameworks in a straightforward manner, making it possible the use of complex frameworks by regular developers. To achieve this goal the Draco-PUC [19] transformational system is used to generate framework instantiation code from code written in the DSLs.

The proposed instantiation process uses the following elements: DSLs, a framework, and a transformational system. The DSLs and the framework gather the domain main concepts. The transformational system is used to map the specification written in the DSLs to framework instantiation code. In this way, a customized application is the combination of the framework with the instantiation code.

Figure *1* presents a flow diagram of this process. As a precondition, this process requires the DSLs definition. The Draco-PUC allows the definition of new languages by the specification of its grammar in a BNF like notation, and a set of transformation rules to generate the instantiation code. The domain designer performs these steps. To generate a specific application a regular user needs to write a specification in DSL and then apply the set of transformation rules to it.

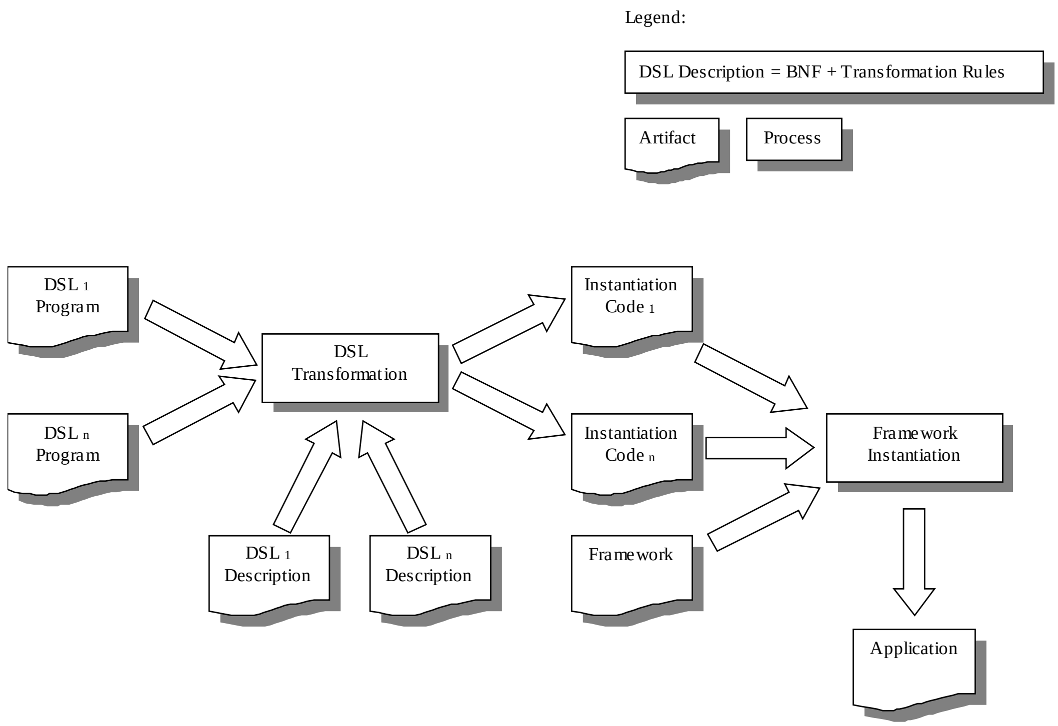

Figure 1. Framework instantiation process

4. Case Study

This section presents a case study about the described instantiation process for the Web-based Education (WBE) domain. We have tried to capture the core functionality of various WBE environments [21, 22, 23, 24, 25, 26] to define an object-oriented framework, called ALADIN [27] [1].

The implementation of this framework is developed in Lua [29] and CGILua [30] and DSLs are used in the framework instantiation. The class structure of the ALADIN framework is now presented using OMT diagrams.

ALADIN allows several institutions to use the same environment simultaneously. Each institution may have several departments. The courses are related to institutions and each course has assigned actors. Each actor may play various roles (student, teacher, or author, for example). A course consists of a selection of services. Each service can be of two different types: internal, which is implemented by the environment, and external, which is a WWW application not implemented by the environment (elsewhere in the Internet). Examples of external

[1] We have used the framework design process described in [28] to define the framework's kernel structure and it's hot-spots.

services are chat applications, CU-SeeMe, e-mail, and list servers. This entire design structure is presented in Figure 2.

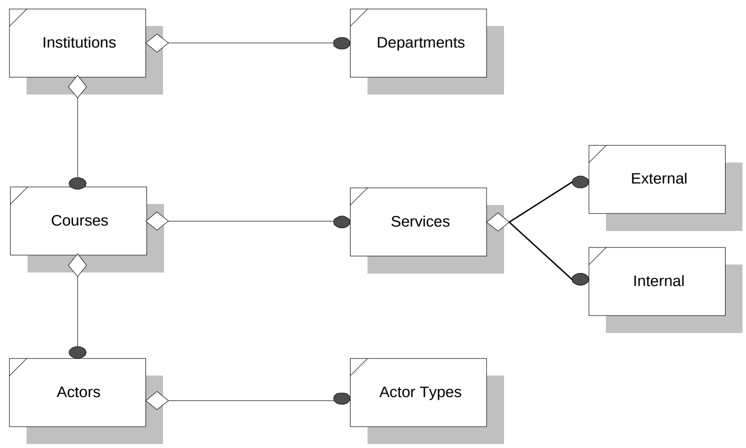

Figure 2. ALADIN OMT class diagram

In ALADIN the final user can modify the visual representation (Interface) for each one of its entities as shown in Figure 3.

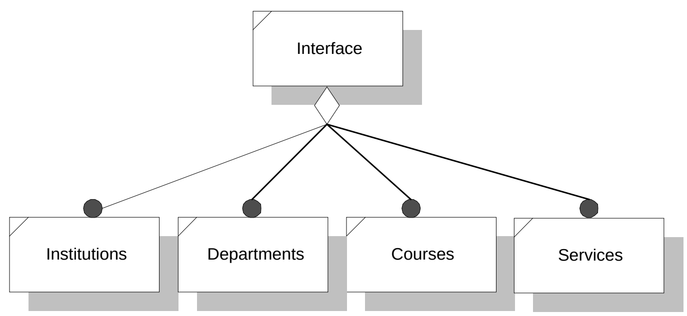

Figure 3. ALADIN interface class structure

Each WBE environment is composed of one WWW site for each type of actor defined. Normally two sites are always present in the existent WBE environments: a learning site and an authoring site. The students use the learning site to attend a specific course, while the authors use the authoring site to create and maintain courses. However, the ALADIN framework allows that others actor types defined in the system have their own WWW site. As an example we could define a site for the monitors and other site for the secretaries. All the sites generated by the ALADIN framework can define many navigational structures. Figure 4 shows the OMT diagrams of the classes that define the navigational structure of the generated sites.

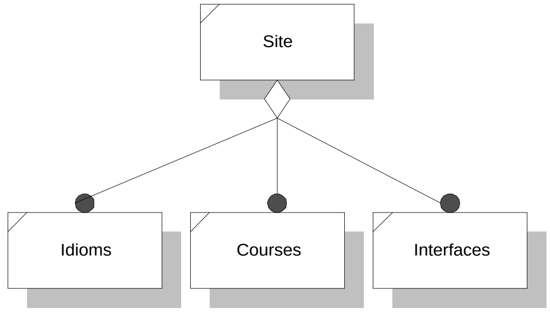

Figure 4. ALADIN site class structure

4.1 ALADIN Hot-spots

The classes institutions, departments, courses, actor types, and services are responsible for accessing their correspondent information in a database system. To provide this functionality gets and sets methods are present in each of the classes. However the attributes that define these entities may vary, and therefore all the access methods are hook methods [5] and their classes are the framework hot-pots.

An example of a usual execution of the framework is shown bellow, where the object course asks for the students actors that attend it.

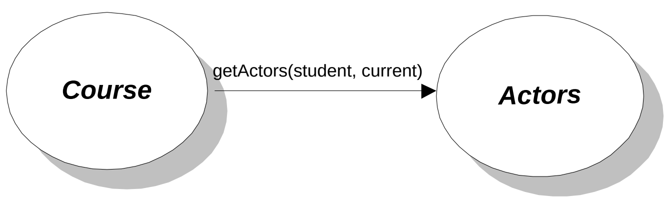

Figure 5. Framework execution example

Since the ALADIN framework allows flexible definitions of the actor's attributes, the method getActors is not defined until the framework is instantiated.

The Interface and Site classes, used to define the visual representation and navigational structure are defined as hot-spots. The Interface class depends on the layout structure (usually defined by an interface designer) while the Site class may have various implementations to the methods that generate the output HTML files.

There are two DSLs used in the ALADIN instantiation process:

1. Educational Language: used in the definition of the educational components (institutions, departments, courses, actors types, and services);

2. Navigational Language: used in the definition of the Language (e.g. English, French), interface (e.g. Background images, buttons), and navigational structure.

4.2 Educational Language

The Educational Language code is transformed into two instantiation codes: one in Microsoft Access Basic, which is used for defining the database structure that will be used by the environment, and other in CGILua, which provides the access methods for accessing the database. Figure 6 illustrates this architecture.

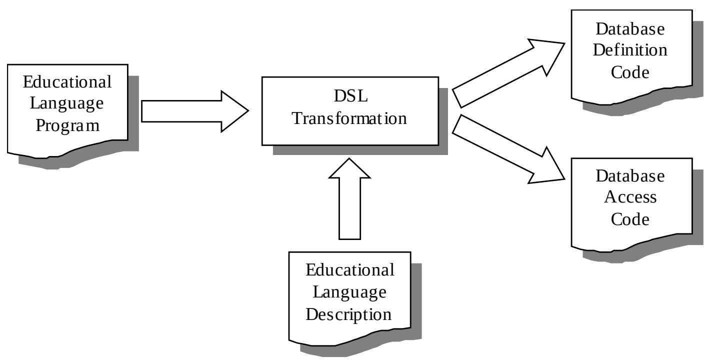

Figure 6. Educational language architecture

The following program code is written in the Educational Language.

```
Institution "PUC-RIO", "Pontificia Universidade Catolica - Rio de Janeiro", "PUC.gif";
Department "CETUC", "TELECOMUNICATIONS", "CETUC.GIF";
Department "CS", "COMPUTER SCIENCE", "CS.GIF";

Actor Type Teacher, "Teacher"
        {       name String;
                description Memo;
                Photo Image; };

Actor Type Student, "Student"
        {       name String;
                description Memo;
                period Integer;
                address String;
                average Real; };
Course
        {       name String;
                code String;
                syllabus Memo;
```

```
                description Memo;
                image Image; };
Service "CourseNews"
        {       news Memo;
                initialDate Date;
                finalDate Date; }
        read = [ Student ]
        write = [ Teacher ];
```

Note that each operator in the language has not only the data used by the environment, but also the meta-information about the structure of these data. The definition of the course attributes in the operator Course is an example. This meta-information is used for the generation of meta-tags in the XML standard as defined in the EDUCOM/IMS project [31]. This approach guarantees that the generated environments are compliant with the IMS platform and assures the content interoperability.

The part[2] of the Microsoft Access Basic code generated by the transformation of the above Educational Language program is shown next.

```
'#################################################
'# Code generated by the EducationLanguage 1.0
'#
'#################################################

Dim wrkDefault As Workspace
Dim rst As Recordset
Set wrkDefault = DBEngine.Workspaces(0)
Dim outputDB As Database
Set outputDB = wrkDefault.CreateDatabase("kernel.mdb", dbLangGeneral, dbVersion30)

'##### Generating Institution Table

Dim Institution As TableDef
Set Institution = outputDB.CreateTableDef("Institution")

Institution.Fields.Append Institution.CreateField("Id",dbLong)

Dim idxInstitution As Index
```

[2] This part of the code creates the relational table for defining institutions and adds a record to it.

```
Set idxInstitution = Institution.CreateIndex("PrimaryKey")
idxInstitution.Fields.Append idxInstitution.CreateField("Id")
idxInstitution.Primary = True
idxInstitution.Unique = True
Institution.Indexes.Append idxInstitution

outputDB.TableDefs.Append Institution
Dim InstitutionNameField As Field
Set InstitutionNameField = Institution.CreateField("Name", dbText,255)
InstitutionNameField.AllowZeroLength = True
Institution.Fields.Append InstitutionNameField
Dim InstitutionDescriptionField As Field
Set InstitutionDescriptionField = Institution.CreateField("Description", dbMemo)
InstitutionDescriptionField.AllowZeroLength = True
Institution.Fields.Append InstitutionDescriptionField
Dim InstitutionImageField As Field
Set InstitutionImageField = Institution.CreateField("Image", dbText,255)
InstitutionImageField.AllowZeroLength = True
Institution.Fields.Append InstitutionImageField

'##### Adding Institution
Set rst = outputDB.OpenRecordset ("Institution",dbOpenDynaset)
With rst
.AddNew
!Id = 1
!Name = "PUC-RIO"
!Description = "Pontificia Universidade Catolica - Rio de Janeiro"
!Image = "PUC.bmp"
.Update
.Close
End With
```

The execution of the complete Microsoft Access basic code generates the Microsoft Access database used by the WBE environment that is being instantiated from the ALADIN framework. As described above, the transformation of the Educational Language also generates the database access methods. The following methods interfaces, written in CGILua, define the methods generated to access course definitions from the previously shown Educational Language program.

```
luaTable = getCourse(name, code, syllabus, description, image)
addCourse (name, code, syllabus, description, image)
updateCourse(oldName, name,  code, syllabus, description, image)
```

```
deleteCourse(name)
```

These methods encapsulate the SQL commands, allowing ALADIN users to generate WBE environments without having any knowledge about manipulating relational databases.

4.3 Navigational Language

The Navigational Language code is transformed into HTML files that define the WBE environment interface and navigational structure. Note that these files also have the embedded CGILua access methods described above. Therefore, the transformation of the Navigational Language generates the WBE environment site structure.

An example of the Navigational Language is provided next.

```
Language "English"
Language "Portuguese"
Text "title1", " English", "Resources"
Text "title1", "Portuguese", "Recursos"
Image "img1", " English", "c:\ing\img.gif"
Image "img1","Portuguese","c:\port\img.gif"

a := template("c:\templates\temp1.html")
b := template("c:\templates\temp2.html")
c := template("c:\templates\temp3.html")

b.next := c
b.previous := a
```

This language provides an operator for defining the languages supported by the environment (Language). The texts and images used by the environment are defined in the various languages. Note that the HTML files are in fact templates, which have special tags for the texts and images. In this example the tag <ALADIN-TEXT>title1</ALADIN-TEXT> would be replaced by the sting “Resources”, if the selected language had been English, and by the string “Recursos”, if the selected language was Portuguese.

The same is valid for the hypertext links (Navigational Structure). In the above example, the tag <ALADIN-LINK>previous</ALADIN-LINK> in the template temp2.html would be replaced by the sting <A HREF="c:\templates\temp1.html">previous</A>, while the tag <ALADIN-LINK>next</ALADIN-LINK> would be replaced by the string <A HREF="c:\templates\temp3.html">next</A>. This approach allows the definition of all the navigational links in a separated file, providing more readability and flexibility.

4.4 ALADIN Application Examples

This section describes how the AulaNet™ [23] authoring and learning sites have been developed using the ALADIN framework. The general structure of the authoring site is shown in Figure 7.

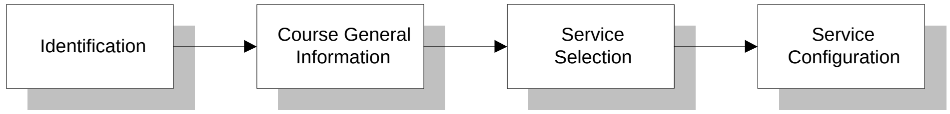

Figure 7. AulaNet authoring site: global view

In the identification step the author has to provide an userid and password to the system. In the course general information step all the basic information about the course must be completed, such as course name, description and syllabus[3]. Then the author has to inform the system about the services necessary for the course. Finally, in the service configuration phase, each Internet service available in the course must be configured. Once all these steps have been completed the course is ready and the learning site can be generated. The structure and HTML code (with embedded CGILua) generated by the ALADIN framework for the resource selection step is shown below.

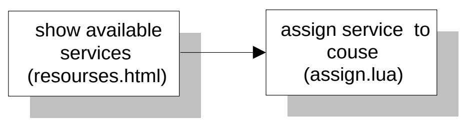

Figure 8. Resource selection: detailed structure

[3] Note that the Course operator (Educational Language) provides the information of what are the fields to be completed in the course general information step.

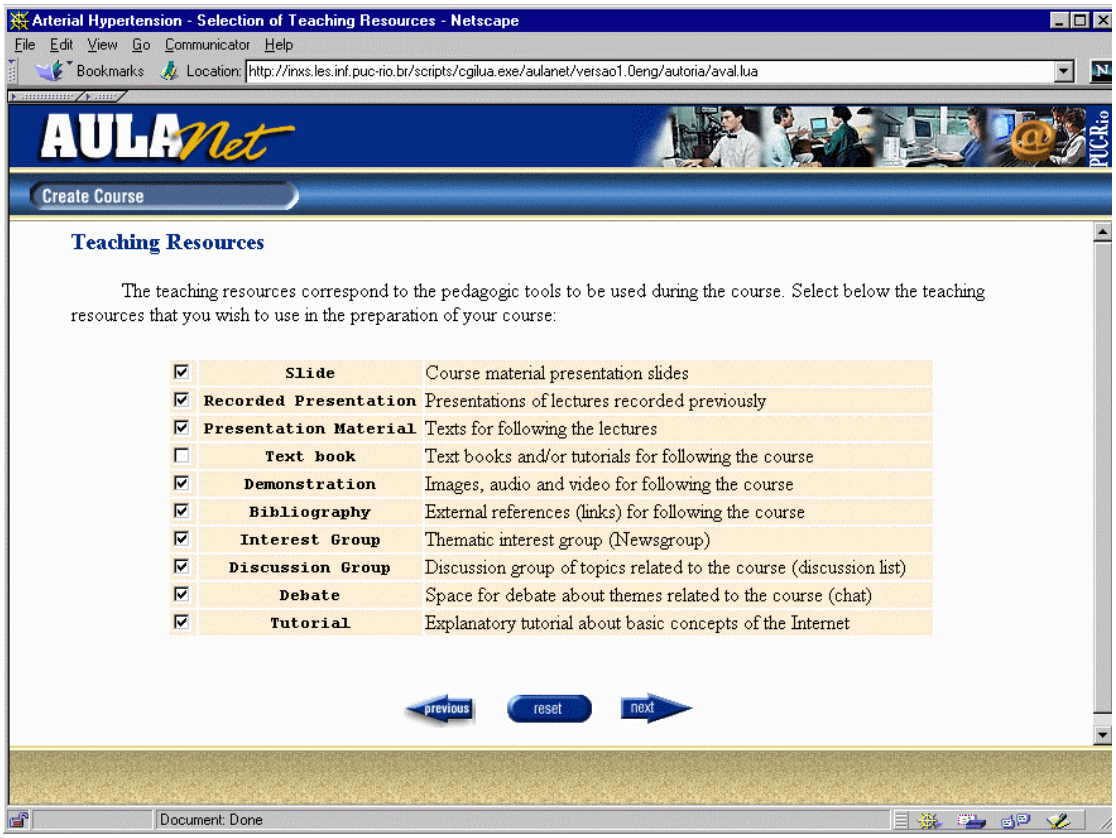

Figure 9. AulaNet authoring site: selection of teaching services

```
****************************************

*              services.html            *

****************************************

<HTML>

<HEAD>

   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">

   <META NAME="Author" CONTENT="Leonardo Moura">

   <META NAME="GENERATOR" CONTENT="Mozilla/4.02 [en] (Win95; I) [Netscape]">

   <TITLE>Show Available Services</TITLE>


<BODY BGCOLOR="#FFFFFF" TEXT="#000000" topmargin=0 leftmargin=0>

<H1> Please select the services you want to use in your course </H1>


<FORM NAME="data" TARGET="_top" METHOD="post" ACTION="assign.lua">


<!--$$

-- Select all the services availabe to the current intitution.

-- The result of this selection is stored in a lua table.

table = getService(currentInstitution)
```

```
while (table ~= nil) do
  write('<td valign=middle>')
  write('<INPUT TYPE="Checkbox" NAME="'..table[0]..'" VALUE="'..table[0]..'">')
  write('</td>')

  moveNext(table)
end

$$-->

<INPUT TYPE="Submit" VALUE="Next">
</FORM>

</BODY>
</HTML>

****************************************
*               assign.lua             *
****************************************
table = getService(currentInstitution)
while (table ~= nil) do
  -- test to check if the resource is select or not
  -- if true add the resource to the course
  if (cgi.table[0] = "ON") then
       write(has(course, table[0]))
  moveNext(table)
end
```

The learning site structure is presented in Figure 10. The student goes through an identification page, providing her e-mail and password. After that, the system presents a list of the courses that the student has been assigned to. The student then selects the course she wants to attend and the system builds and presents the course main page. The course main page contains all the general information about the course, as well as the list of services available. Once the student selects the desired service a frame that implements it is presented.

The HTML code (with embedded CGILua) generated by the ALADIN framework for the list of courses step is shown below.

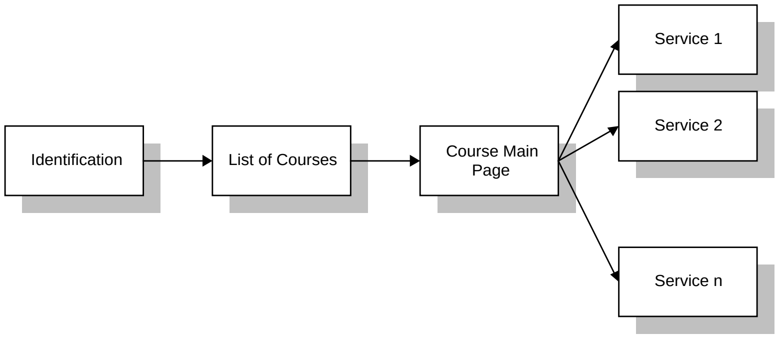

Figure 10. AulaNet learning site: global view

```
****************************************
*            courselist.html           *
****************************************
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
   <META NAME="Author" CONTENT="Leonardo Moura">
   <META NAME="GENERATOR" CONTENT="Mozilla/4.02 [en] (Win95; I) [Netscape]">
   <TITLE>Show List of Cousers</TITLE>

<BODY BGCOLOR="#FFFFFF" TEXT="#000000" topmargin=0 leftmargin=0>
<H1> Please select the course you want to attend now</H1>

<FORM NAME="data" TARGET="_top" METHOD="post" ACTION="coursemain.html">

<!--$$
table = getActorsCourses(currentActor, student)

while (table ~= nil) do
  write('<td valign=middle>')
  write('<INPUT TYPE="Radio" NAME="'..table[1]..'" VALUE="'..table[1]..'">')
  write('</td>')

  moveNext(table)
end

$$-->

<INPUT TYPE="Submit" VALUE="Attend">

</FORM>
```

```
</BODY>
</HTML>
```

Note that the generated HTML and CGILua files are very simple and easy to be understood since they use an embedded high-level language, which is the Database Access Code.

5. Related Work

In [32, 33] a similar work for software libraries is described. The motivation for their work is that end users often do not make effective use of libraries, because most of them lack the expertise to properly identify and compose the routines appropriate to their application. They argue that in domains with mature subroutine libraries, one can greatly improve the productivity and quality of software engineering by automating the effective use of those libraries. We view our work as an enhancement of this approach for frameworks. A main difference is related to the technique to generate the application, in [32, 33] the code is generated based on a deductive approach [34] using constructive theorem proving. In our technique the code is generated using a transformation system with a generative component concept [35]. In this system simple generative components can be composed to generate complex transformations.

The wizards in existent integrated development environments (IDE) [36, 9] can be seen as a primitive kind of software generator. In these environments the user interacts with a sequence of dialog boxes in order to generate code. The main drawback is that all generators are hard-wired in these systems, and it is not possible to define new abstractions.

6. Support Environment

Figure 11 describes a support environment for the framework instantiation process, which is based on a configurable syntax driven editor. The editor is used to help the framework users to write the specification in the appropriate DSLs. For doing that it needs to be customized for each particular DSL. This customization is made through the uses of a domain specific language for syntax driven editor domain. The Draco-PUC transformational system is then used to generate the editor configuration code. This approach allows domain experts to be able to write their specifications in an editor specialized for their domain, which seems to be an appropriate interface.

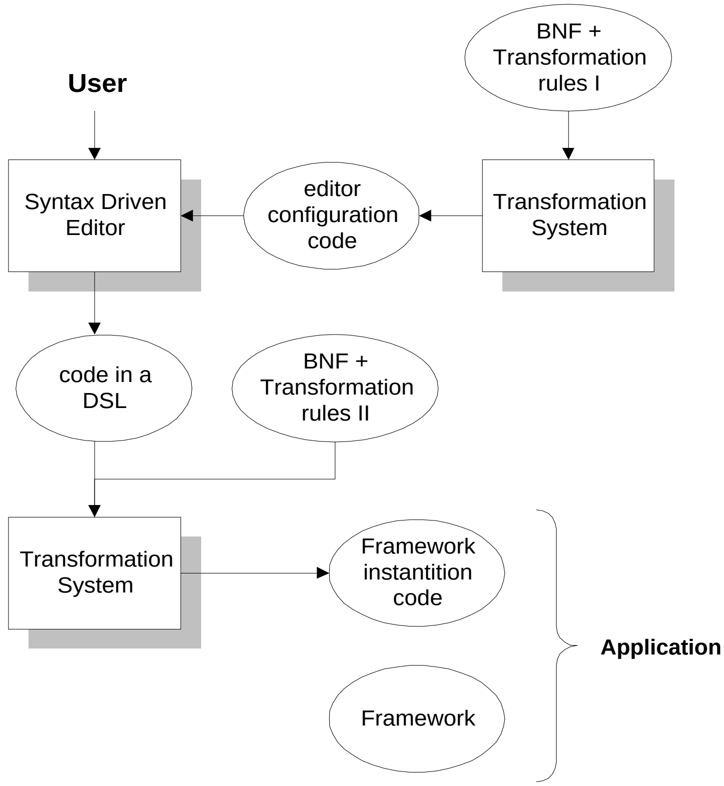

Figure 11. Framework instantiation support environment

7. Conclusion

This paper presents an approach to integrate frameworks with domain specific languages (DSL). We argue that DSLs allows the domain expert to formalize the specification of a software solution immediately without worrying about implementation decisions and the framework complexity. To implement this approach a transformational system (Draco-PUC) is used to transform a DSL specification in framework instantiation code.

This approach seems to be useful and can be used in other situations where the frameworks are very difficult to use. It will be tested in others important and innovative domains, in which we also expect to contribute for the design and development of real frameworks (electronic commerce and computational biology).

It is important to note that DSLs can be transformed into other DSLs, thus creating a domain network, in a way similar to that described in [18], providing an easy implementation path for new DSLs.

We are now working on a derivation approach for domain-specific languages based on problem theory [37]. This seems to be an interesting approach not yet described in literature.

Since the WBE domain is still not completely understood the need for a framework that supports fast development of alternative WBE environments by non-programmers is a desirable goal. One advantage of our approach is that we can experiment with different environments while minimizing development costs. Another advantage is that teachers and education researchers can develop their own environments, with little help from software engineers.

The environments generated by the ALADIN framework are compatible with the new standards described in the EDUCOM/IMS project [31], allowing their integration with others IMS servers through the use of meta-data o accomplish the content interoperability.

A new version of AulaNet™ (http://www.les.inf.puc-rio.br/aulanet) [23] is now being developed through the use of ALADIN. This experiment has two main purposes: the development of a more flexible version of AulaNet™ and the further validation of the ALADIN framework.

REFERENCES

1. GAMMA, E., HELM, R., JOHNSON, R. E., and VLISSIDES, J., "Design Patterns, Elements of Reusable Object-Oriented Software", (Addison-Wesley, 1995).

2. JOHNSON, R., "Frameworks = (Components + Patterns)", *Communications of the ACM*, October 1997, **40**(10).

3. ROBERTS, D. and JOHNSON, R., "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks" in "Pattern Languages of Program Design 3", (Addison-Wesley, 1997).

4. WIRFS-BROCK, R. and JOHNSON R., "Surveying current research in object-oriented design", *Communications of the ACM*, 1990, **33**(9).

5. PREE, W., "Design Patterns for Object-Oriented Software Development", (Addison-Wesley, 1995).

6. KICZALES, G. Kiczales, DES RIVIERES, J., and BOBROW, D. G., "The Art of Mataobject Protocol", (MIT Press, 1991).

7. JOHNSON, R. and OPDYKE, W. F., "Refactoring and aggregation", In Object Technologies for Advanced Software, First JSSST International Symposium, 1993, pp. 264-278.

8. CASAIS, E.: "An incremental class reorganization approach", In ECOOP'92 Proceedings, volume 615 of Lecture Notes in Computer Science, 1992, pp. 114-132.

9. VISUAL C++ 4.0 USER'S GUIDE, Microsoft Inc., 1995.

10. WEINAND, A., GAMMA, E., and MARTY, R., "ET++ An object-oriented application framework in C++", Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings, San Diego, CA, 1988, pp. 46-57.

11. CLINIO, A. L.: "VIX - Um Framework para Suporte de Objetos Visuais Interativos", Tese de Mestrado, Departamento de Informática, PUC-Rio, 1996 (in Portuguese).

12. SIMONYI, C., "The Death Of Computer Languages, The Birth of Intentional Programming", Technical Report MSR-TR-95-52, September 1995.

13. LEVY, C., COWAN, D., LUCENA, C. J., GATTASS, M., and FIGUEIREDO, L. H., "IUP/LED: A Portable User Interface Tool", *Software Practice and Experience* (accepted for publication).

14. SCHMID, H. A., "Systematic Framework Design by Generalization", *Communications of the ACM*, October 1997, **40**(10).

15. BELL, J.: "Software design for reliability and reuse: A proof-of-concept demonstration", In TRI-Ada '94 Proceedings, ACM, November 1994, pp. 396-404.

16. DIJKSTRA, E. W.: "The humble programmer", *Communications of the ACM*, October 1972, **15**(10).

17. HUDAK, P., "Building Domain-Specific Embedded Languages", *Computing Surveys*, ACM, 1996, **28A**(4).

18. NEIGHBORS, J. M., "The Draco Approach to Constructing Software from Reusable Components", *IEEE Transactions on Software Engineering*, September 1984, **10**(5).

19. LEITE, J. C. S. P., SANT'ANNA, M., and FREITAS, F. G., "Draco-PUC: a Technology Assembly for Domain Oriented Software Development"; Proceedings of the 3rd IEEE International Conference of Software Reuse, 1994.

20. CORDY, J. and CARMICHAEL, I.: "The TXL Programming Language Syntax and Informal Semantics", Technical Report, Queen's University at Kingston, Canada, 1993.

21. COWAN, D.: "An Object-Oriented Framework for LiveBOOKs", Technical Report, CS-98, University of Waterloo, Ontario, Canada, 1998.

22. LEARNINGSPACE, <http://www.lotus.com/home.nsf/welcome/learnspace>.

23. LUCENA, C., FUKS, H., MILIDIU, R., MACEDO, L., SANTOS, N., LAUFER, C., RIBEIRO, M., FONTOURA, M. F., NOYA, R., CRESPO, S., TORRES, V., DAFLON, L., and LUKOWIECKI, L., "AulaNet™ - An Environment for the Development and Maintenance of Courses on the Web", Proceedings of ICEE'98, 1998 (to appear).

24. SIMON FRASER UNIVERSITY, <http://virtual-u.cs.sfu.ca/vuweb/>.

25. UNIVERSITY OF BRITISH COLUMBIA, <http://homebrew.cs.ubc.ca/webct/>.

26. VIRGINIA COMMONWEALTH UNIVERSITY, <http://views.vcu.edu/wcb/intro/wcbintro.hml>.

27. COWAN, D., CRESPO, S., FONTOURA, M. F., LUCENA, C. J., and MOURA, L. M.: "ALADIN: An Architecture for Learningware Applications Design and Instantiation", MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to ICSE'99).

28. ALENCAR, P., COWAN, D., CRESPO, S., FONTOURA, M. F., and LUCENA, C. J. Lucena: "Formalizing a Framework Design Process", MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to Journal of Systems and Software).

29. IERUSALIMSCHY, R., FIGUEIREDO, L. H., and CELES, W., "Lua an extensible extension language", *Software Practice and Experience*, 1996, **26**(6), pp 635-652.

30. IERUSALIMSCHY, R., BORGES, R., and HESTER, A. M.: "CGILua A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages", SBES'97 (Simpósio Brasileiro de Engenharia de Software), 1997.

31. IMS, <http://www.imsproject.org/specs.html>.

32. PRESSBURGER, T. and LOWRY, M., "Automatic Domain-Oriented Software Design using Formal Methods", Software Systems in Engineering, Energy-Sources Technology Conference and Exhibition, Houston, Texas, 1995, pp. 33-42.

33. STICKEL, M., WALDINGER, R., LOWRY, M., PRESSBURGER, T., and UNDERWOOK, I., "Deductive Composition of Astronomical Software from Subroutine Libraries", in 12th Conference on Automated Deduction, Nancy, France, June 1994.

34. MANNA, Z. and WALDINGER, R., "Fundamentals of Deductive Program Synthesis", *IEEE Transactions on Software Engineering*, 1992, **18**(8), pp. 674-704.

35. SANT'ANNA, M., LEITE, J. C. S. P, DO PRADO, A. F., "A Generative Approach To Componentware", CBSE'98 - International Workshop on Component Based Software Engineering, Kyoto, Japan, 1998.

36. DELPHI USER'S GUIDE, Borland Inc., 1995.

37. TURSKI, W. and MAIBAUM, T. S. E., "The Specification of Computer Programs", (Addison-Wesley Publishing Company, 1987).