# ALADIN: an Architecture for Learningware Applications Design and Instantiation

**Marcus Felipe M. C. da Fontoura, Leonardo M. de Moura, Sérgio Crespo, and Carlos José P. de Lucena**
Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil
e-mail: [mafe, moura, crespo, lucena]@inf.puc-rio.br
PUC-RioInf.MCC34/98 Outubro, 1998

ABSTRACT

This paper presents an abstract software Architecture for Learningware Applications Design and Instantiation (ALADIN), which is based on a conceptual model for Web-based education (WBE) environments. ALADIN is designed through a combination of architectural patterns and implemented through an object-oriented framework and domain-specific languages (DSLs). The DSLs are used for the framework instantiation to help the framework user to create a specific application in an easier way, without worrying about implementation decisions and remaining focused on the problem domain. The specification written in the DSLs are then translated to the framework instantiation code using a transformational system. Utilization examples showing how WBE environments can be implemented with ALADIN are also shown.
KEY WORDS: Web-based Education (WBE) environments, conceptual model for WBE, architectural patterns, components, object-oriented frameworks, domain-specific languages, framework instantiation.

RESUMO

Este artigo apresenta uma arquitetura abstrata para desenvolvimento e instanciação de aplicações de learningware (ALADIN), que é baseada em um modelo conceitual para educação baseada na Web (WBE). O design de ALADIN é baseado em uma combinação de patterns arquiteturais e a implementação é feita através de um framework orientado a objetos e linguagens de domínio (DSLs). A especificação escrita em linguagens de domíno é transformada no código de instanciação do framework através de um sistema transformacional. Exemplos de utilização mostrando como ambientes para WBE podem ser implementados com ALADIN também são mostrados.
PALAVRAS CHAVES: Ambientes para educação baseada na Web (WBE), modelo conceitual para WBE, patterns arquiteturais, componentes, frameworks orientados a objetos, linguagens de domínio, instanciação.

## 1. INTRODUCTION

Education and training based on the Web uses the WWW as a means of publishing didactic material, applying tutorials, applying exams and tests and communicating with students. The Web can also be used as a vehicle of communication for the presentation of distance lectures (multimedia conferences).

A learning process, at any level of instruction and training, normally incorporates the following action, as stated by the Internet2 project (http://www.internet2.edu):

- Establishing the objectives of the learning process;
- Finding and revising (or creating) instructional material (e.g.: diagnostic tools, textbooks, courseware, tests, etc.);
- Assessing students' level of knowledge;
- Assigning appropriate material to students;
- Defining the form of access students have to components/modules;
- Revising and following-up students' progress and intervening when necessary;
- Providing and managing communication between student and instructor and between students themselves (synchronous as well as asynchronous);
- Assessing the learning process;
- Preparing reports of the results of the learning process.

Support for Web-based Education (WBE) is under development by many different groups, and new ideas and models are presently under evaluation. The possibility of rapidly building prototypes of new WBE environments will be of significant assistance to educators who wish to experiment with possible educational approaches based on the Web.

This paper presents an abstract software architecture, called ALADIN, in which domain experts can develop applications in an easy way, without the need of programming expertise. This goal was accomplished by the development of an object-oriented framework [Johnson 1997; Pree 1995] and domain-specific languages (DSLs) [Hudak 1996] for WBE. The framework architecture is based on patterns [Buschmann *et. al.* 1996] and components [Johnson 1997].

The framework and the DSLs presented here were defined based on a conceptual model for WBE. The derivation of this conceptual model is presented in [Alencar *et. al.* 1998] and was based on six different existent WBE environments.

## 2. A CONCEPTUAL MODEL FOR WBE ENVIRONMENTS

The conceptual model presented here is based on components. Figure 1 illustrates the model while each of the components is presented next.

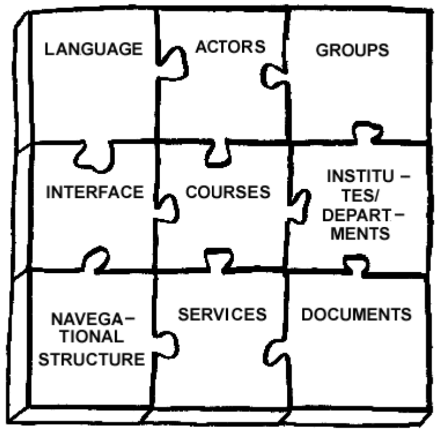


Figure 1. Conceptual model based on components.

- Courses: this component allows the creation and maintenance of courses. The content of each course is controlled by the components services and documents;
- Actors: are the ones that interact with the environment. Actors can belong to different roles, such as student, teacher, and secretary;
- Services: provide the functionality needed by the course. Services can have different types, like administrative services (course agenda, bulletin board, etc.), communication services (chat, e-mail, etc.), teaching services (transparency, references on the Web, etc.) and evaluation services (quizzes, self-evaluation, etc.). Services can also be internal or external. Internal services are implemented by the environment while the external ones are WWW applications not implemented by the environment (elsewhere in the Internet). Examples of external services are chat applications, CU-SeeMe, e-mail, and list servers;
- Documents: are the artifacts used by the internal services (external services do not use this component). As an example, the service “Course News” uses a document that has a field containing the news that is going to be published;
- Groups: this component allows cooperative work, letting actors interact to complete tasks;
- Institutions and Departments: this component allows the environment customization for various institutions and departments that will use it;
- Language: provides support for various languages (English, French, Portuguese, etc.);
- Interface: allows the interface customization;
- Navigational Structure: allows the navigational structure customization.

Each of these components work as a hot-spot [Pree 1996; Schmid 1997] of WBE environments. As an example, the AulaNet™ environment [Lucena *et. al.* 1998] does not implements the capabilities of customizing navigational structure and interface, thus the component model of this environment is the one presented in Figure 2.

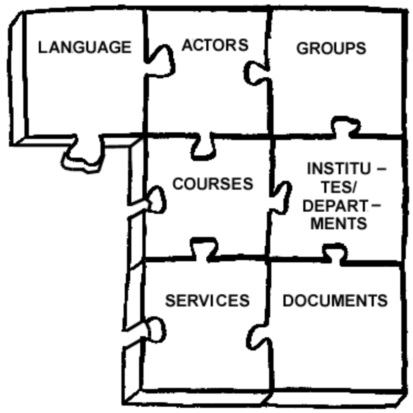


Figure 2. AulaNet component model.

A work comparing various WBE environments through this component-based conceptual model can be seen in [Crespo *et. al.* 1998].

# 3. ALADIN DESIGN MODEL

This section describes the ALADIN design model, showing how the components above described interact. Architectural patterns [Buschmann *et. al.* 1996] are used to define the framework fundamental structure [Shaw and Garlan 1996]. ALADIN was developed in CGILua [Ierusalimschy *et. al.* 1996, 1997].

## 3.1 Layered Architecture

The Layers pattern [Buschmann *et. al.* 1996] was used to decompose the ALALIN design structure separating orthogonal tasks in different layers. Figure 3 illustrates the three layers defined. The persistence layer is responsible for managing all the database accesses. The components layer encapsulates the components described in Section 2. Finally, the application layer is a components layer client and provides the functionality required by the WBE application.

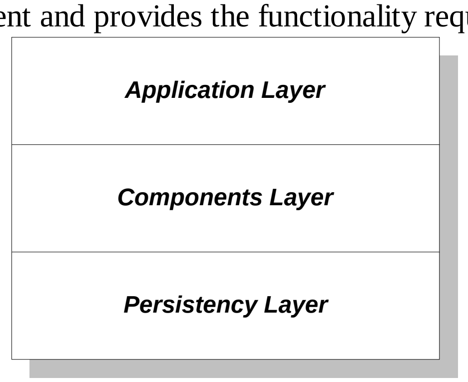


Figure 3. ALADIN Layered Architecture.

The layering division was implemented through the Facade design pattern [Gamma *et. al.* 95], which provides a unified interface to each layer. Figure 4 illustrates this approach.

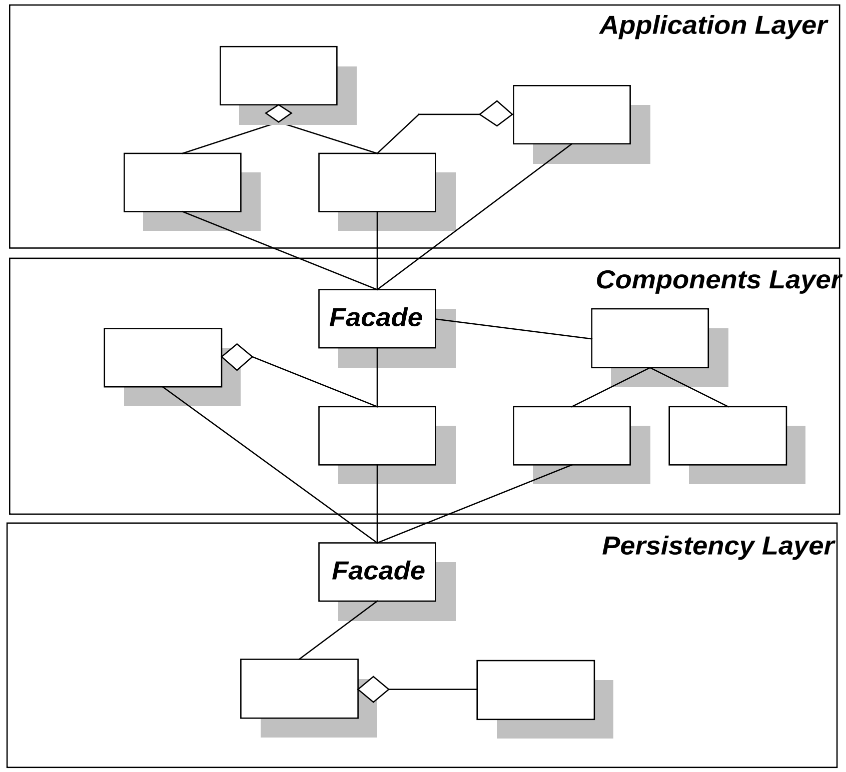

Figure 4. Implementing a layered architecutre through the facade pattern.

The following sub-sections detail each of the three layers, beginning from the lower level of abstraction layer.

## 3.2 Persistence Layer

The persistency layer is responsible for the storage of all the system's objects. It hides the database structure and localization from the other layers, avoiding that changes in the database do affect their implementation.

This layer utilizes a single class, called StorageService, which is responsible for all the system's database access methods. When a storage operation is invoked an object of this class (StorageObject) is created to handle the request. All the storage operations are performed in two steps:

- In the first step StorageObject consults a database table[1] that has the necessary information for storing ALADIN objects. This table, called DefinitionTable, has the information of how the objects have to be stored, that is, their storing device location and structure. Note that StorageObject does not know how to store the systems objects, this information is provided by the database itself. StorageObject only knows the DefinitionTable structure and localization;
- StorageObject then uses the retrieved information to conclude the operation.

[1] The word table is used because the actual ALADIN implementation utilizes a relational database system. However, this access structure works with any persistency model.

This access strategy is transparent to the other layers. Figure 5 shows the persistency layer design structure through OMT [Rumbaugh *et. al.* 1991] like class diagrams.

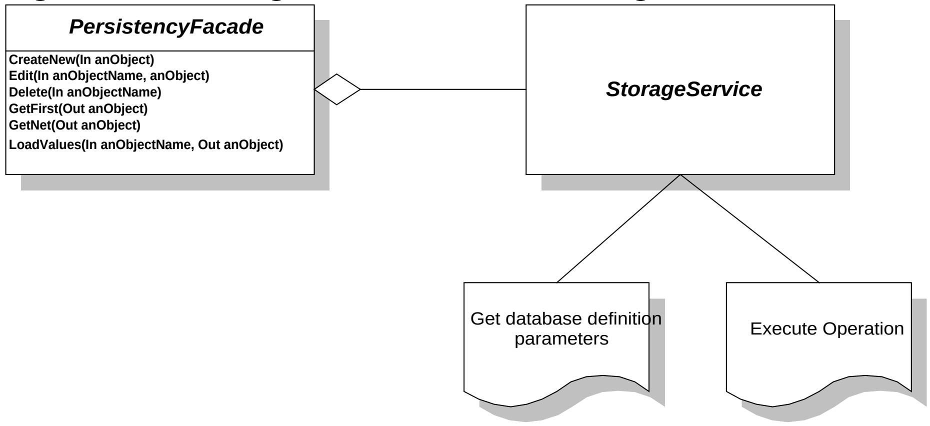


Figure 5. Persistency layer class diagram.

## 3.3 Components Layer

This layer implements the relationship between the conceptual model components described in Section 2. This layer, however, does not handle the navigational structure, interface, and language components, which will be discussed in the application layer.

ALADIN components can be classified in two groups: base level and meta level components. The base level components implement the application logic while the meta level components are used to provide run time configuration capabilities. A meta-object protocol (MOP) is used to redefine component structure at run time. This structure is based in the Reflection pattern [Buschmann *et. al.* 1996] and is illustrated in Figure 6.

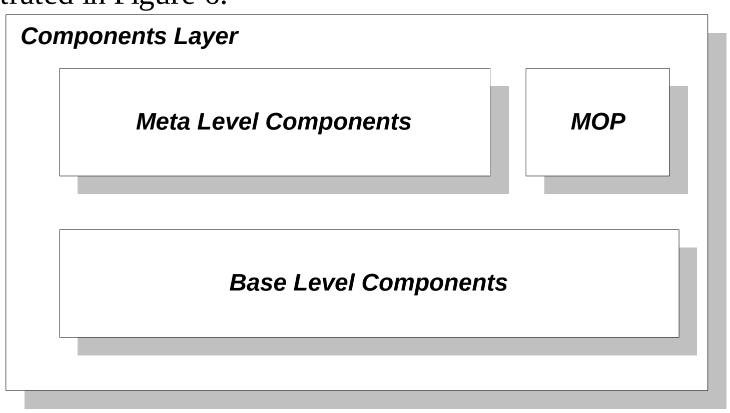


Figure 6. Reflection pattern.

The actors, documents, and interface components are meta level components, while the others are base level ones. The class structure of the base level components layer 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. A course consists of a selection of services. Each service can be of two different types: internal and external. Internal services use documents. Actors can belong to groups. This entire design structure is presented in Figure 7.

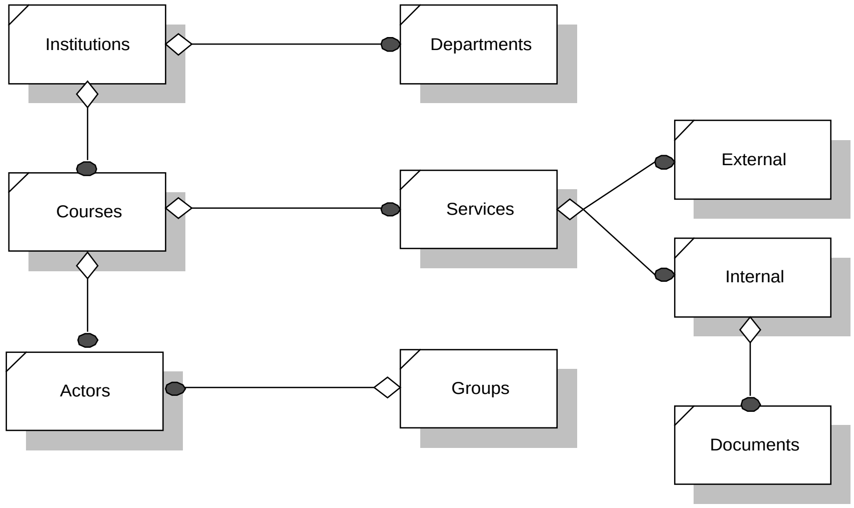


Figure 7. Base level components class diagram.

The actors component is a meta level component because each actor may play various roles (student, teacher, or author, for example). Thus, the structure of the component may vary for each different actor type (or role). ActorTypes is a meta level class responsible for providing this information to the base level Actors class. Every time a request is sent to the Actors class it has to ask the ActorTypes class which attributes belong to that actor type. The ActorFields base level class provides the attribute instances. This design structure is presented in Figure 8.

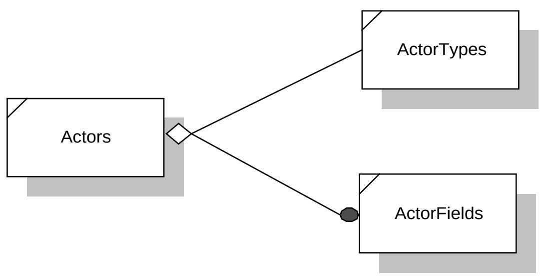


Figure 8. Actors component detailed structure.

The documents component follows the same structure since different document types can be defined to be used in different internal services. The relationship between the classes Documents, DocumentTypes, and DocumentFields is similar to the one presented in Figure 8 for the actors classes.

The classes Actors and Documents are responsible for accessing their correspondent information. However the attributes that define these entities may vary, and therefore their methods may have different implementations and their classes are the framework hot-pots [Pree 1996]. Note that the meta-object protocol classes are responsible for providing the appropriate methods for restructuring the components at run time, and for this reason, they have to change their definition in the DefinitionTable described in Section 3.2.

An example of a usual execution of the framework is shown in Figure 9, where the object Course asks for the students actors that attend it.

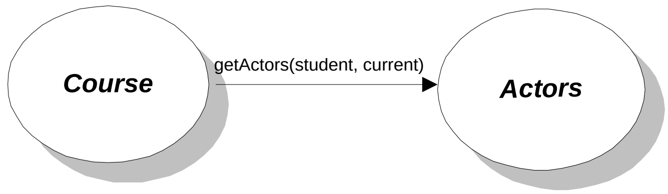


Figure 9. Framework execution example.

Since the ALADIN framework allows flexible definitions of the actor's attributes, the method getActors is not defined until the framework is instantiated. To help this instantiation process a DSL, called Educational Language, is used to configure the components layer hot-spots.

### 3.3.1 Educational Language

The Educational Language is a domain specific language used to configure the ALADIN components layer. The Draco-PUC transformational system [Sant'anna et. al. 1998; Leite et. al. 1994] is used to transform the code written in Educational Language into CGILua code to complete the hot-spot methods. The transformation of the Educational Language code also generates a program in Microsoft Access Basic that is used for defining the database structure that will be used by the generated WBE environment. Figure 10 illustrates this structure.

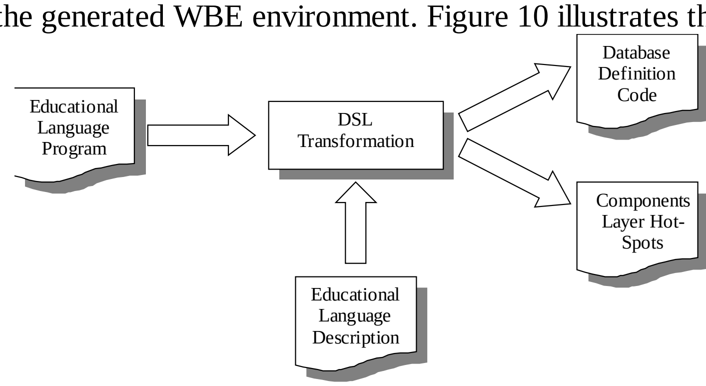


Figure 10. Educational language architecture.

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 student and teacher attributes in the operator Actor Type 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 (http://www.imsproject.org). 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
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
```

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

```
'##### 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 access methods that complete the components layer hot-spot methods. The following methods interfaces, written in CGILua, define the methods generated to the Course component 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)
```

## 3.4 Application Layer

The application layer is responsible for defining the interface, language, and navigational structure components for the generated WBE applications.

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. A DSL, called Navigational Language, is used to define de ALADIN application layer.

### 3.4.1 Navigational Language

The Navigational Language code is an extension of the Script language code, which generates HTML files that define the WBE environment interface, language, and navigational structure. The Script language is described in Section 4. The generated HTML 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 (Interface) 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 string “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 string <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. The Script Embedded Language

An extension programming language called Script was built to be used as a flexible configuration language for the Spider environment [Moura *et. al.* 1998]. It is based on the Lua language developed in PUC-Rio [Ierusalimschy *et. al.* 1996]. This language simplifies the customization work, avoiding the implementation of simple tools in Java language. Another salient point is that the customization process becomes more dynamic, so there is no need to restart the system when a new tool is created or modified. The Script language is implemented in Java. It is worth noting that, being an extension language, Script must be embedded in a Java program, and may be used in other systems, different from the Spider environment, such as the ALADIN framework in this case. The Script language features may be summarized in the following way.

## 4.1 Procedural programming

All statements in Script are executed in an environment. The environments are created using the Java language. Each environment keeps its own global variables and functions. Each environment can be manipulated by Script code or by the embedding program (i.e. Java code). A unit of execution is composed of statements and function declarations.
Script supports the conventional set of statements: blocks, assignments, control structures (while, if and repeat until), function calls, local declarations, and transparent access to Java objects.

## 4.2 Dynamically typed language

Script is a dynamically typed language. Variables do not have types, only values do. All values carry their own type. Therefore, there are no type definitions. There are the following basic types in Script: Null, Integer, Float, String, Function, Array (Table), JavaObject, JavaMethod, JavaClass, and JavaClassMethod. The types are organized in a Java class hierarchy that has as root the class ScriptObject. This root class defines the required interface for a Script type. This approach is similar to the one implemented in the Python language [Rossum 1993].
Functions are considered first-class citizens in Script. This mean that functions can be stored in variables and passed as argument to other functions. The type Array (Table) implements an

associative array that can be indexed using any kind of value. Therefore, this type can be used to represent records, symbol tables, sets, etc.
The types JavaObject, JavaMethod, JavaClass and JavaClassMethod are used to provide transparent access to Java code.

### 4.3 Extensible

It is very easy to extend the Script language with new features, due to its object-oriented design. This design allows the user to exploit the power of object-oriented techniques to make the language implementation adjustable and flexible, using standard techniques of sub-classing and specialization. Our approach is based on meta-object protocols ideas [Kiczales *et. al.* 1991].
For instance, the user can add new primitive types creating new subclasses of the class ScriptObject. Actually, we use this approach to implement the transparent access to the Java code.

### 4.4 Transparent access to Java code (no stub code is necessary)

Script code can access Java code in a transparent way. This kind of feature is possible due to the meta-object protocol of the Script language and the reflexive features of the Java language. We can summarize our approach in the following way:

- We defined the primitive types JavaObject, JavaClass, JavaMethod, and JavaClassMethod. These types wrap the access to the Java code.
- When an access to Java code is requested, the above wrappers convert the argument values (Script values) to the corresponding Java values. After converting the arguments, the Java code is accessed using the Java reflexivity API.
- As Script is a dynamically typed language, type errors can happen when calling Java code. We cope with this problem generating a Script exception. If the Java code generates an exception, it is converted to a Script exception.

### 4.5 Other features

Script has a lot of others features borrowed from the Lua language: fallbacks; anonymous and nested functions; reflexivity; coercion; etc.

### 4.6 A Script Language Example

In this section, we a simple example of Script language code. More complex examples can be found in [Moura *et. al.* 1998]. This example shows how Java objects can be called from the Script language. This piece of code writes the following message in the file “arq.txt”**.**

> *hello world!*
> *testing script language.*
> *end.*

```
FileWriter  = class java.io.FileWriter;
PrintWriter = class java.io.PrintWriter;

output = new PrintWriter (new FileWriter ("arq.txt"));
output.println("hello world!");
output.println("testing script language.");
output.println("end.");
output.close();
```

# 5. ALADIN UTILIZATION EXAMPLES

This section describes how the AulaNet™ [Lucena *et. al.* 1998] authoring and learning sites are developed using the ALADIN architecture. The general structure of the authoring site is shown in Figure 11.

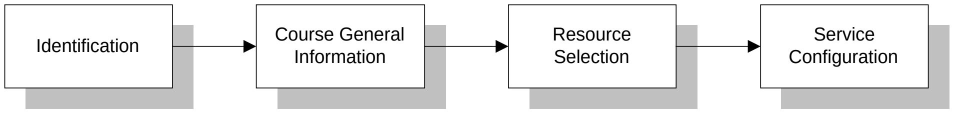


Figure 11. AulaNet authoring site: global view.

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 resources necessary for the course. Finally, in the service configuration phase, each Internet service available in the course must be configured. Once all these steps are completed the course is ready and the learning site is generated. The structure and HTML code (with embedded CGILua and code) generated by the ALADIN architecture for the resource selection step is shown below.

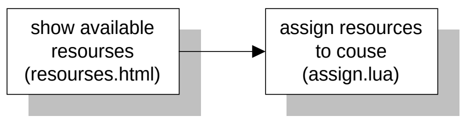


Figure 12. Resource selection: detailed structure.

```
****************************************
*             resources.html           *
****************************************
<HTML>
<HEAD>
<TITLE>Show Available Resources</TITLE>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" topmargin=0 leftmargin=0>

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

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

<!--$$
-- select all the resources availabe to the current intitution and
-- department. The result of this selection is stored in a lua table

table = querylang("getResource(currentInstitution, currentDepartment)")

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">
```

---

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

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

****************************************
*               assign.lua             *
****************************************
table = querylang("getResource(currentInstitution, currentDepartment)")
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
      wbelang("has(course, table[0])")
  moveNext(table)
end
```

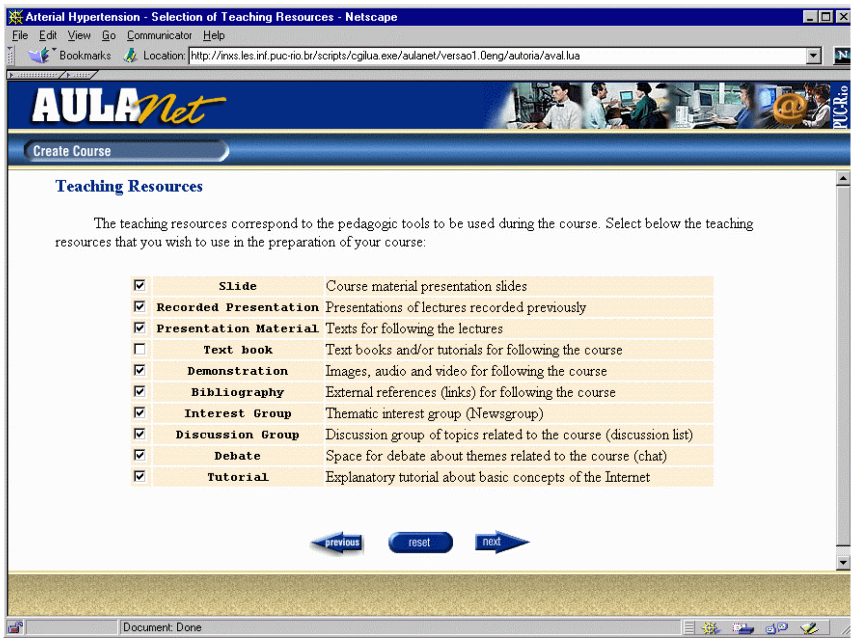


Figure 13. AulaNet authoring site: selection of teaching services.

The learning site structure is presented in Figure 14. The student passes trough an identification page, providing his or her e-mail and password. After that the system presents a list of the courses that the student is assigned to. The student then selects the course he or 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 code) generated by the ALADIN architecture for the list of courses step is shown below.

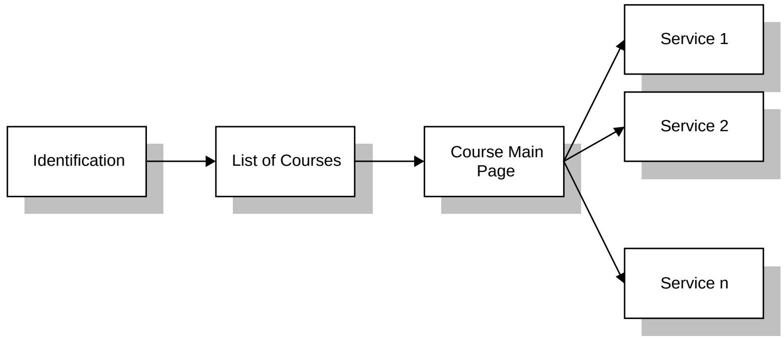


Figure 14. AulaNet learning site: global view.

```
****************************************
*            courselist.html           *
****************************************
<HTML>
<HEAD>
<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">

<!--$$
-- select all the resources availabe to the current intitution and
-- department. The result of this selection is stored in a lua table
table = querylang("getActorInCourse(currentStudent, 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 embedded CGILua code is used to access the components layer facade, which is completed implemented as Lua functions.

## 6. RELATED WORK

The Microsoft Windows DNA (http://www.microsoft.com/dna) is an architecture for developing WWW applications through the use of Microsoft tools. Our approach, however, is not focused on any kind of WWW application but just the learningware domain. The EDUCOM/IMS project (http://www.imsproject.org) defines a model for learningware applications and implements a software prototype. The environments generated by the ALADIN architecture are compatible with the new standards described in this project, allowing their integration with others IMS servers through the use of meta-data to accomplish the content interoperability.

From the software engineering perspective the use of domain-specific languages to configure the framework can be compared to the work presented in [Stickel *et. al.* 1994; Pressburger and Lowry 1995] for software libraries. 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 [Stickel *et. al.* 1994; Pressburger and Lowry 1995] the code is generated based on a deductive approach [Manna and Walddinger 1992] using constructive theorem proving. In our technique the code is generated using a transformation system with a generative component concept [Sant'anna *et. al.* 1998]. In this system simple generative components can be composed to generate complex transformations.

## 7. CONCLUSIONS AND FUTURE WORK

Since the WBE domain is still not completely understood the need for an architecture 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.

A new version of the AulaNet™ environment [Lucena *et. al.* 1998] is now being developed by the use of ALADIN. This experiment has two main purposes: the development of a more flexible version of AulaNet™ and the validation of the ALADIN architecture (http://www.les.inf.puc-rio.br/aulanet).

**REFERENCES**


Alencar, P., D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena (1998), "Using Viewpoints to Derive a Conceptual Model for Web-Based Education Environments", Technical Report MCC17/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, Brazil (also submitted to Journal of Systems and Software, http://www.les.inf.puc-rio.br/~mafe).

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal (1996), *Pattern-Oriented Software Architecture: A System of Patterns,* John Wiley & Sons.

Crespo, S., M. F. Fontoura, and C. J. Lucena (1998), "Using a Conceptual Model Compatible with the EDUCOM/IMS Platform to Compare Web-based Educational Environments", In *Proceedings of the Brazilian Symposium on Education and Computer Science 1998*, Fortaleza, Brazil (in Portuguese).

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995), *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Hudak, P. (1996), "Building Domain-Specific Embedded Languages", Computing Surveys 28A, 4.

Ierusalimschy, R., L. H. Figueiredo, and W. Celes (1996), "Lua an extensible extension language", *Software Practice and Experience* 26, 6, 635-652.

Ierusalimschy, R, R. Borges, and A. M. Hester, (1997), "CGILua A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages", In *Proceedings of the Brazilian Symposium on Software Engineering 1997*, Fortaleza, Brazil.

R. Johnson (1997), "Frameworks = (Components + Patterns)", *Communications of the ACM* 40, 10, 39-42.

Kiczales, G., J. des Riviéres, D. G. Bobrow (1991), *The Art of the Metaobject Protocol*, MIT Press.

Leite, J., M. Sant'anna, and F. Freitas (1994), "Draco-PUC: a Technology Assembly for Domain Oriented Software Development", In *Proceedings of the 3rd IEEE International Conference of Software Reuse*.

Lucena, C., H. Fuks, R. Milidiu, L. Macedo, N. Santos, C. Laufer, M. Ribeiro, M. Fontoura, R. Noya, S. Crespo, V. Torres, L. Daflon, and L. Lukowiecki (1998), "AulaNet™ - An Environment for the Development and Maintenance of Courses on the Web", In *Proceedings of the International Conference on Engineering Education 1998,* Rio de Janeiro, Brazil.

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

Moura, L., C. Lucena, and A. von Staa (1998), "The Spider Environment", *Software Paractice and Experience,* (accepted for publication).

Pree, W. (1995), *Design Patterns for Object-Oriented Software Development*, Addison-Wesley.

Pree, W. (1996), *Framework Patterns*, Sigs Management Briefings.

Pressburger, T. and M. Lowry (1995), "Automatic Domain-Oriented Software Design using Formal Methods", In *Proceedings of the Software Systems in Engineering, Energy-Sources Technology Conference and Exhibition*, Houston, Texas, 33-42.

Rossum, G. van (1993), "An introduction to Python for UNIX/C programmers", In *Proceedings of the NLUUG najaarsconferentie,* Dutch UNIX users group. (ftp://ftp.cwi.nl/pu b/pyt hon/nluug- paper.ps ).

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen (1991), *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Clifs.

Sant'anna, M., J. Leite, and A. do Prado (1998), "A Generative Approach To Componentware", In *Proceedings of the International Workshop on Component Based Software Engineering 1998*, Kyoto, Japan.

Schmid, H. A. (1997), "Systematic Framework Design by Generalization", *Communications of the ACM* 40, 10, 48-51.

Shaw, M. and D. Garlan (1996), *Software Architecture – Perspectives on an Emerging Discipline*, Prentice Hall.

Stickel, M., R. Walddinger, M. Lowry, T. Pressburger, and I. Underwook (1994), "Deductive Composition of Astronomical Software from Subroutine Libraries", In *Proceedings of the 12th Conference on Automated Deduction*, Nancy, France.