Using Viewpoints to Derive a Conceptual Model for Web-Based Education Environments

Paulo Alencar*, Donald D. Cowan*, Sergio Crespo♣, Marcus Felipe M. C. da Fontoura♣, and Carlos José P. de Lucena♣

*Computer Science Group, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

♣Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil
(e-mail: [alencar, dcowan]@csg.uwaterloo.ca, [crespo, mafe, lucena]@inf.puc-rio.br)

PUC-RioInf.MCC17/98 Abril, 1998

ABSTRACT

This paper proposes a conceptual model for Web-Based Education (WBE) environments that is implemented through an object-oriented framework. We also define a domain-specific language (DSL), which is used to instantiate the framework, as part of the model. The process of defining this model is based on viewpoints and on our viewpoint-based development method for object-oriented frameworks. The domain analysis was developed through the study of various WBE environments, which were used as viewpoints of this type of environment. A description of the development method is presented to clarify its application in the elicitation and design processes. The object models of the analyzed environments are also presented. The resulting conceptual model is currently being implemented in a visual programming language and will be used for the generation of different WBE environments.

KEY WORDS: object-oriented frameworks, viewpoints, framework development method, domain-oriented software development, domain specific languages, Web-based educational environments.

RESUMO

Este artigo apresenta um modelo conceitual para ambientes de educação baseados na Web (WBE), que é implementado através de um framework orientado a objetos. Nós também definimos como parte do modelo uma linguagem de domínio (DSL), que é utilizada na instanciação do framework. O processo de definição do modelo é baseado em viewpoints e no método para desenvolvimento de frameworks orientados a objetos baseado em viewpoints. A análise de domínio foi desenvolvida através do estudo de vários ambientes para WBE, que foram utilizados como viewpoints. Uma descrição do método de desenvolvimento é apresentada para clarificar sua aplicação nos processos de elicitação e design. Os modelos de objetos dos ambientes analizados também são apresentados. O modelo conceitual resultante está sendo implementado em uma linguagem de programação visual e será utilizado na geração de vários ambientes para WBE.

PALAVRAS CHAVES: frameworks orientados a objetos, viewpoints, método de desenvolvimento de frameworks, desenvolvimento de software baseado em domínios, linguagem específica de domínio, ambientes para educação baseados na Web.

1. INTRODUCTION

Support for Web-based education 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.

The goal of our present work is to define a precise conceptual model for WBE environments. A version of this conceptual model is already defined and is now being implemented as a meta-environment in a visual programming language [6]. We believe that this environment will allow non-programmers to develop WBE environments quickly and easily.

This paper explains the derivation of the conceptual model. The analysis process was based on six different WBE environments and uses a viewpoint-based framework development method [2]. In this approach, each one of the environments is a different viewpoint of WBEs, which are then unified to generate the kernel of the object-oriented framework for WBE.

In [20] Roberts and Johnson state that "Developing reusable frameworks cannot occur by simply sitting down and thinking about the problem domain. No one has the insight to come up with the proper abstractions." They propose the development of concrete examples in order to understand the domain. Our strategy is quite similar, analyzing each one of the viewpoints as a concrete example and deriving the final framework from the analysis.

We have also defined a domain-specific language, which was incrementally improved during the domain analysis process. This language is now being used in the implementation of the WBE conceptual model, to assist in the framework instantiation and evolution process [6].

This paper is divided as follows. Section 2 describes our viewpoint-based development approach and how it was applied to the WBE domain; the object models resulting from the domain analysis are presented in Section 3; Section 4 shows the complete definition of our conceptual model; and the conclusions and future work are presented in Section 5.

2. A MODELING APPROACH BASED ON VIEWPOINTS AND A DOMAIN SPECIFIC LANGUAGE

A framework [14] is defined as generic software for an application domain. A framework provides a reusable semi-finished software architecture that allows both single building blocks, and the design of the sub-system to be reused.

Our approach to framework development is based on the idea that any framework design can be divided into two parts: the kernel sub-system design and the application-specific sub-system design [2]. The kernel sub-system design is common to all the applications that the framework may generate, and the application-specific sub-system design describes the different characteristics of each application that can be supported by the framework. The application-specific sub-system uses the methods and information provided by the kernel sub-system and usually extends it.

The first step of the modeling approach is to discover the kernel and the application-specific structures. We develop framework design diagrams based on viewpoints, where a viewpoint defines a possibly different use of the framework (Figure 1).

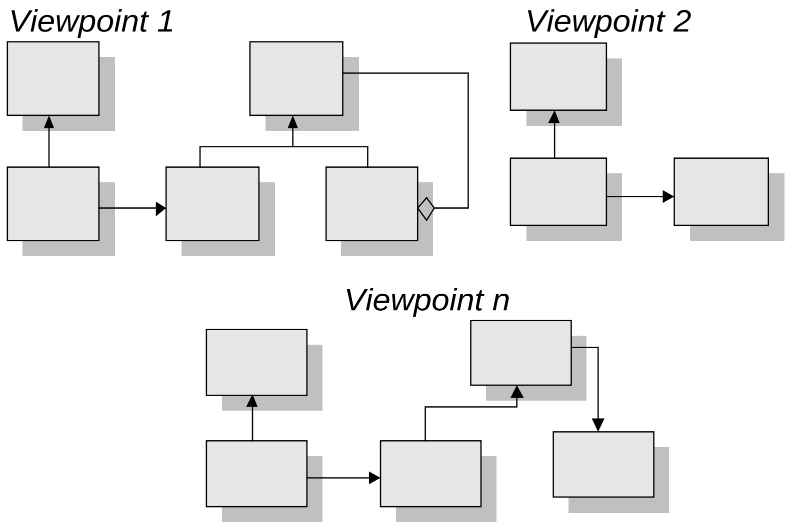


Figure 1. Different Viewpoints of the same Framework

Once we have all the relevant viewpoints, we can discover the kernel structures by analyzing the design representations to produce a resulting design representation that reflects a structure that is common to all chosen viewpoints (we informally call these common structures the "*unification*" of the viewpoints). This part of the design approach is based on a domain-dependent semantic analysis of the design diagrams to elicit the common features of the classes and relationship. The common part is the kernel sub-system. For example the kernel for the viewpoints in Figure 1 is shown in Figure 2.

In order to help in the unification step of the process we use domain-specific languages (DSL) [4, 10] that reflect the viewpoint structure. The basic idea is that each viewpoint can be represented by a set of declarative rules, in a PROLOG-like language style. Furthermore the analysis of each one of this sets of rules can lead to a high level language that expresses the domain entity requirements. We can than map this language into OMT class structures. Our belief is that it is easier to interpret declarative textual languages than object-oriented diagrams. In fact both approaches are complementary and the use of diagrams and DSL can reduce errors, since the comparison between them is a way of checking the model. The derived language can then be used in framework instantiation and evolution. Because the viewpoints can be expressed in this way we should be able to produce tools that assist in the unification process.

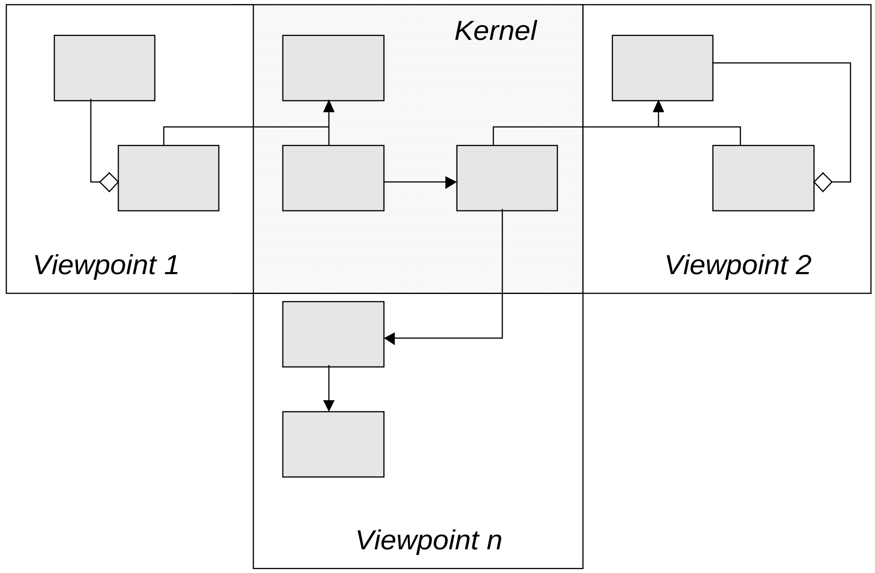


Figure 2. Unification of the Viewpoints

The elements that are not in the kernel are the ones that vary for each application, and depend on the use of the framework. These elements define the framework hot-spots [19, 21] that must be adapted to each related application. The hot-spots are used to define the structure of a application-specific sub-system. Hot-spots can be identified through the same semantic analysis procedure used to identify the kernel.

Each hot-spot represents an aspect of the framework that may have a different implementation for each instantiation. A hot-spot can be characterized as a relation between an application-specific object and a kernel object in which the application-specific object uses the services of the kernel and adapts them to its own view of the services. We use the views-a relation [3] to specify all the hot-spots in the system, and we generate a design diagram based on the views-a relationship [3].

The views-a relationship and its realization through *Abstract Design Views (ADV)* [7] is a concept that promotes loose coupling between modules. The views-a relationship is maintained between two objects, when one object called the “view” monitors or “views” the states of another related object (viewed). The related object is unaware of the existence of the viewing object.

The framework hot-spots can be instantiated through views [7] that represent the application-specific objects. In this case, views can be characterized as components [14] that may be plugged into the hot-spots. The actual implementation of each views-a relation is made by the selection of a design pattern [8]. This selection process is guided by views-a cards [2], which provide a systematic way for selecting design patterns based on the views-a relationship properties [3]. The process of using the views-a relationship and instantiating it through design patterns is described in [2,3].

In the WBE domain we have analyzed each environment as a different viewpoint and then used semantic unification to generate the framework structure. We have also used the DSL to model each one of the analyzed environments as a set of rules. As a result of the analysis process we obtain a conceptual model for WBE which is characterized by an OO framework and a DSL. We are now implementing this model using the Spider [18] visual programming environment by producing a high level architecture in which non-programming experts are able to develop WBE environments through black-box composition [6].

Since the objective of our comparison of WBE environments is the definition of a conceptual model, we were not interested in a feature by feature analysis of the environments (a summary of the functionality of the various environments is presented in Appendix B). We only need to analyze the core entities of the WBE domain. A useful concept adopted throughout the elicitation process was the concept of services. We define a service as a capability provided by the environment. Examples of services are: discussions groups, course news, quizzes, and bulletin boards. Also note that we are not interested in navigational structures. These points will be made clearer as we explain each of the models.

3. SOME EXISTING MODELS FOR WBE

In this section we present the models analyzed in the WBE domain elicitation process. Each of these models was considered as a different viewpoint of an object-oriented WBE framework. Except for AulaNet and LiveBOOKs, we do not know the exact object-model of the analyzed environments. The models presented here are specified by examining the use of these environments.

3.1 AulaNet

AulaNet [16] is a WBE environment developed in the Software Engineering Laboratory, at PUC-Rio. AulaNet is a result of our experience in teaching different courses using Web technology [23, 13, 24]. AulaNet which is presented in OMT diagrams, is based on the following principles:

1. The courses must emphasize interaction, making the student participate in the learning process ("learningware") [12].
2. The author of a course does not need to be an Internet expert.
3. The courses must provide the resources available in a conventional classroom as well as those available from the Web environment. The idea is to allow a smooth transition between the traditional and Web-based approach to teaching.
4. Reuse of existing material available in a digital form must be possible. For example, existing files could be imported through a file upload operation.

AulaNet allows several institutions to use the same environment simultaneously. Each institution may have several departments. The courses are related to institutions and may be either composite or single. Courses are also classified by a degree of difficulty (course levels).

Each course has assigned actors and groups. Groups are composed of actors and other groups. Each actor may either be a student, a teacher, or an author. The student attends a course, the author creates a course, and the teacher delivers a course. A student can also be assigned to a student level. A course consists of a selection of resources. Teaching, evaluation, and administrative resources may be incorporated into a course by the author.

Each resource may be mapped into one or more Internet services. As an example, we can use a teaching resource, which is a "Lesson Text," implemented by Internet services as a text file upload or a URL reference. Each service can also use one or more external tools, such as chat, CU-SeeMe, e-mail, or list servers. The entire design structure is presented in Figure 3.

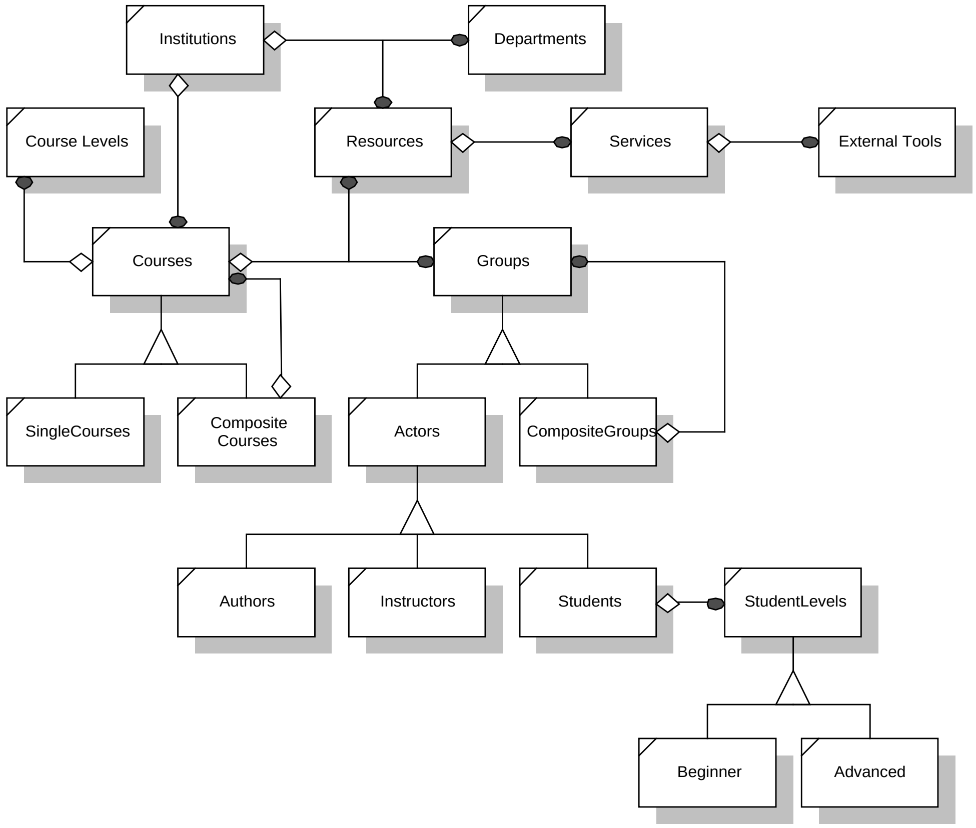


Figure 3. AulaNet OMT Class Diagram

In AulaNet the final user can modify the visual representation (Interface) for each one of its entities as shown in Figure 4.

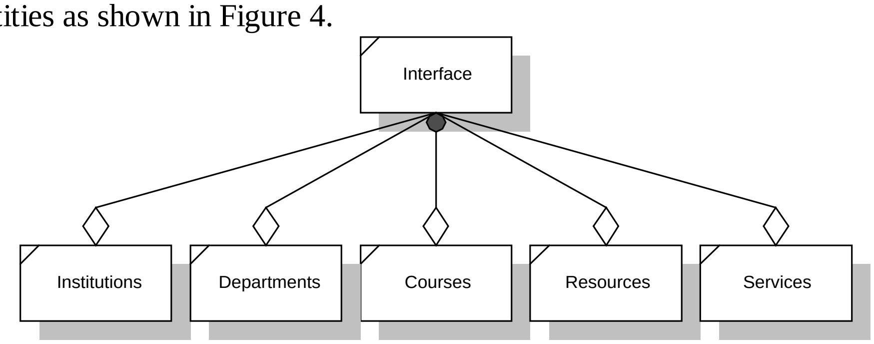


Figure 4. AulaNet Interface Class Structure

Each WBE environment is composed of two sites: a learning site and an authoring site. The students use the learning site to attend a specific course, while the authoring site is used by the authors to create and maintain courses. Both can define many navigational structures. In this comparison we are not going to consider navigation We will define for each of the analyzed environments, as well as for the final framework, the structure of both sites as hot-spots that are implemented by the classes Authoring Site and Learning Site.

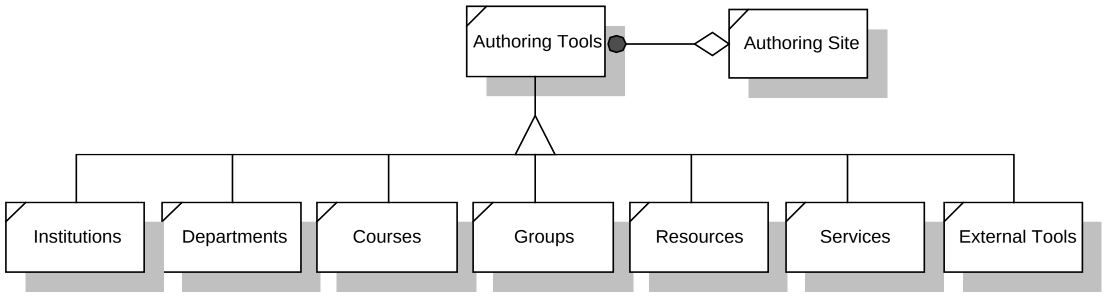


Figure 5. AulaNet Authoring Site

Figures 5 and 6 show the OMT models for the AulaNet authoring and learning sites, respectively. Notice that the learning site has to cope with multiple language requirements or idioms, since a single institution may want to have their courses published in several languages.

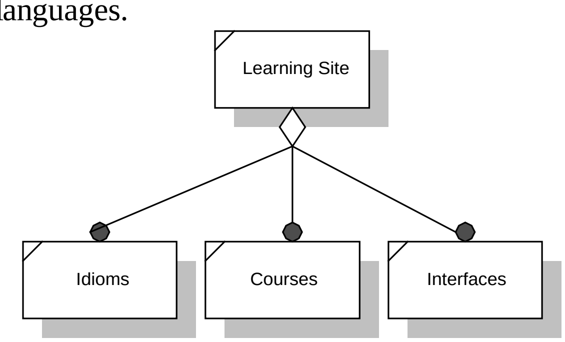


Figure 6. AulaNet LearningSite

3.2 LiveBOOKs

The LiveBOOKs [5] distributed learning/authoring environment is a computer-based teaching/learning and authoring system that supports learning and authoring activities. The environment is designed for ease of use by all actors. We now describe the LiveBOOK class structure.

A LiveBOOK class consists of an aggregation of a LiveBOOK content class and this simple structure is illustrated in Figure 7. The LiveBOOK class has methods to support browsing, updating, generation of the Table of Contents (TOC), and providing the current location of the browsing operation in the LiveBOOK. This latter method is used by the LiveBOOK control class to determine which controls should be present at any point in the LiveBOOK. In other words the object model supports context-sensitivity. The TOC method will be used by the user interface to display the Table of Contents when the LiveBOOK is displayed on the screen.

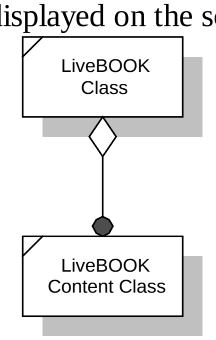


Figure 7. An OMT description of a LiveBOOK class

A LiveBOOK content class illustrated in Figure 8 is an aggregation of one or more LiveBOOK chapter classes.

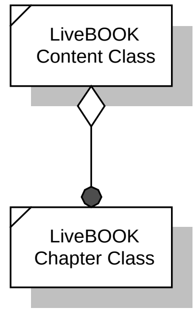


Figure 8. LiveBOOK Content Class Structure

A LiveBOOK Chapter class shown in Figure 9 is an aggregation of zero or more LiveBOOK Fragment classes and zero or more LiveBOOK Component classes. A LiveBOOK Component class is an aggregation of zero or more LiveBOOK Fragment classes and zero or more LiveBOOK Section classes. This definition is recursive in that a LiveBOOK Section class is an aggregation of a LiveBOOK Component class. This relationship is also shown in a LiveBOOK Fragment class illustrated in Figure 10, where the different types of fragment classes inherit from the fragment base class. The fragment base class is an abstract class because the methods on the class depend on its specific type. For example, the display method would depend on the fragment being displayed on the screen. A fragment can consist of text, a binary large object (blob), an image map, a hyperlink source or hyperlink destination, and a hyperlink reference class. A blob is usually a graphic, a picture, or an audio or video clip. An image map is a graphic or picture with a map to allow accompanying hyperlinks. A hyperlink reference class activates an external program.

The user interface of the LiveBOOK is important because it controls the variability of the presentation and in fact the format and style of the educational material. There are many possible user interfaces. This LiveBOOK user interface consists of three windows: a contents window, a table of contents (TOC) window, and a control window. The contents window displays the contents of a subsection and allows the user to interact with the individual LiveBOOK fragments in that subsection. The controls are context sensitive, and the control window represents the various controls that are accessible when a specific contents window is displayed.

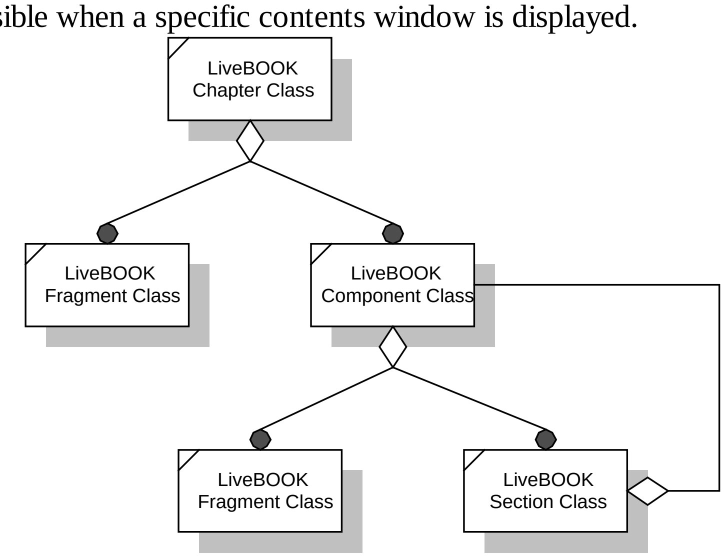


Figure 9. LiveBOOKs' Chapter Class OMT Description

This structure presented for LiveBOOKs represents the implementation of the LiveBOOKs service, and every course developed in LiveBOOK has this service as default.

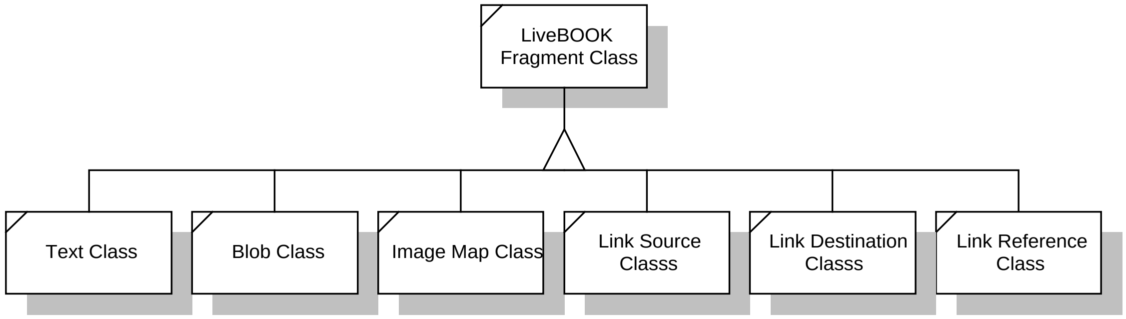


Figure 10. An OMT Description of a LiveBOOK Fragment Class

3.3 Web Course in a Box

Web Course in a Box (WCB) [26] is a course creation and management tool for Web-based or Web-assisted delivery of instruction. It enables creation of basic course pages such as syllabus, class schedule, personal home pages, as well as interactive Web functions such as discussion forums and self-correcting exercises. Authoring is done using a Web browser and does not require any knowledge of HTML or other technical skills.

The WCB class structure (Figure 11) is very similar to the one presented in Figure 3. The main difference is that WCB does not use the concept of resource; the Internet services are directly selected by the author. Note that unlike AulaNet, WCB does not support the concepts of institutions, departments, composite courses, and groups.

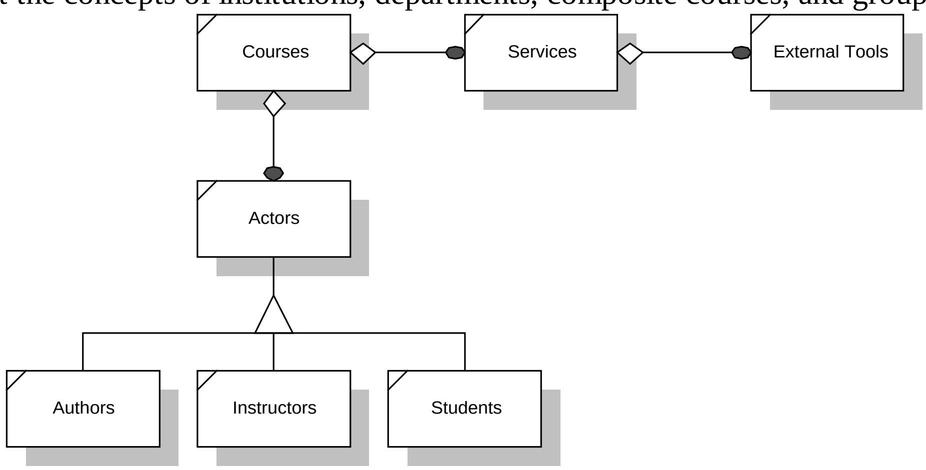


Figure 11. WCB OMT Class Diagram

In relation to visual representation, WCB allows changes only to the course appearance. This structure is presented in Figure 12.

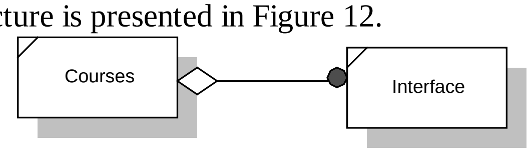


Figure 12. WCB Interface Class Structure

Both the authoring and learning sites are handled in WCB in the same way: as hot-spots that are implemented by the Authoring Site and Learning Site classes. Note that the multiple language requirements are not supported by WCB.

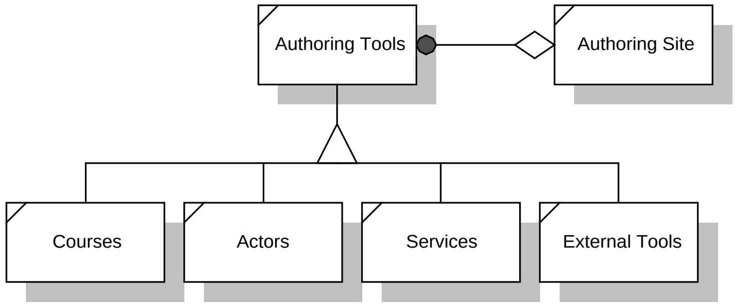


Figure 13. WBC Authoring Site

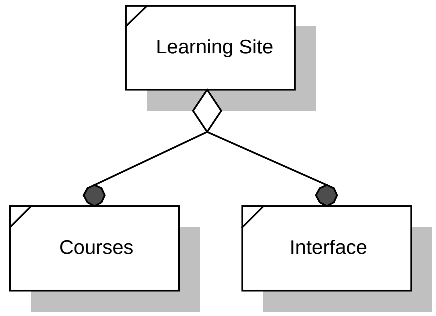


Figure 14. WCB Learning Site

3.4 WebCT

WebCT [25] is a tool that facilitates the creation of sophisticated WBE environments. It requires minimal technical expertise on the part of the developer of the educational material and on the part of the student.

The course developer or designer provides the content of a course. Interactivity, structure, and educational tools are provided by WebCT. WebCT, as well as allowing the incorporation of a set of educational tools in a course, also allows the manipulation of layout and look of the course. WebCT was developed at the University of British Columbia to facilitate the production of various courses for the university [9].

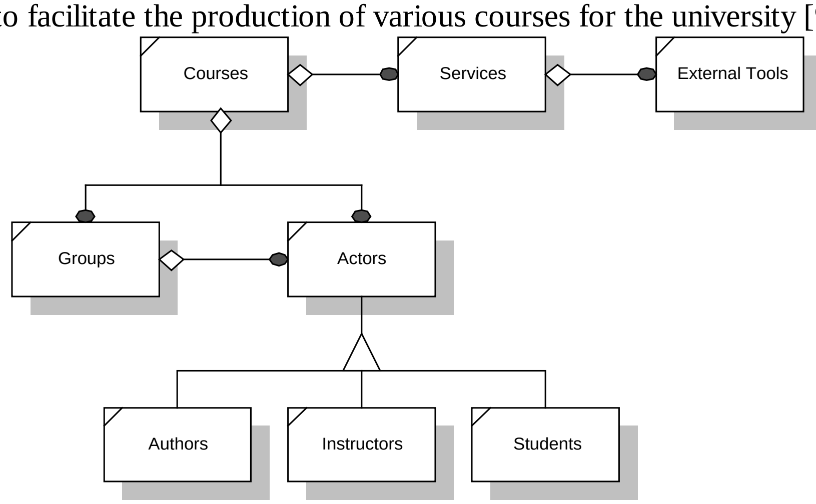


Figure 15. WebCT Class Structure

The WebCT (Figure 15) class structure is very similar to WCB (Figure 11) but it also supports the concept of groups. In relation to visual representation (Interface), WebCT allow changes not only in the course appearance, but also in the services and external tools (Figure 16). The authoring and learning site also have the same structure presented in WCB (Figures 13 and 14).

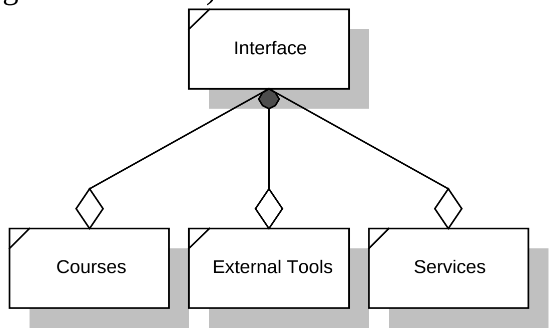


Figure 16. WebCT Interface Class Structure

3.5 LearningSpace and Virtual-U

These two environments are discussed together because they have similar class structures. Lotus Education and IBM are responsible for the research and development of Lotus LearningSpace [15], an educational technology with supporting services for distance education.

The LearningSpace and Virtual-U [18] class structures are essentially the same as the one presented in Figure 13. However, they also introduce the concepts of documents and tasks (Figure 17). The services are based on documents, and each document may have various assigned tasks. In LearningSpace, documents and tasks can be classified in categories (Figure 18).

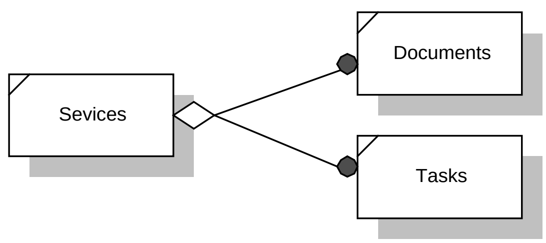


Figure 17. LearningSpace and Virtual-U Services Class Structure

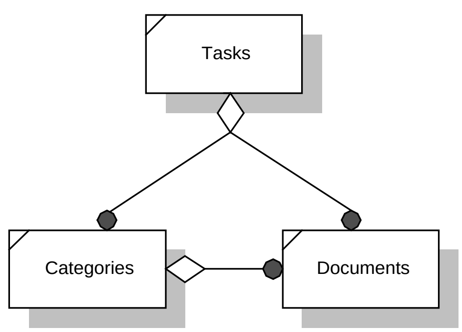


Figure 18. LearningSpace Tasks, Categories, and Documents

4. THE CONCEPTUAL MODEL: A FRAMEWORK AND A DSL FOR WBE

Based on the analysis of the previous WBE environments we have defined an object-oriented framework and a domain-specific language (DSL). We now describe this conceptual model for WBE environments. Section 4.1 shows the complete framework model while Section 4.2 describes the language and its use.

4.1 Object-Oriented Framework

Using the viewpoint based approach we have tried to capture the core functionality of all the analyzed WBE systems, and define an object-oriented framework [14] to handle all requirements for this domain. We have used the viewpoint semantic unification approach [2] to define the framework's kernel structure and its hot-spots.

Figure 19 illustrates (in an abstract way) how we used each one of the analyzed environments as a viewpoint of the final framework. The basic idea was to identify a kernel that could provide all the functionality required for generating, at least, the six environments.

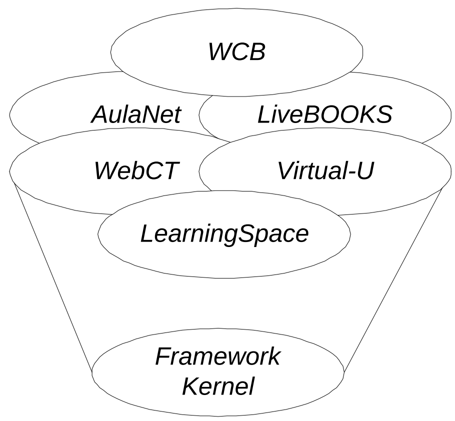


Figure 19. Viewpoints Unification (Abstractly)

The unification used to generate the relation between services and documents is presented, as an example, in Figure 20. AulaNet, WCB, and WebCT (grouped here as Viewpoint1) do not support the document concept. LearningSpace and Virtual-U are represented as Viewpoint 2 since they have similar ways of handing documents. Finally, LiveBOOKs is represented as Viewpoint 3. In LiveBOOKs documents are represented by LiveBOOKs Classes (Figure 7).

Analyzing each one of these viewpoints we see that the framework kernel has to support the definition of documents to be able to generate applications similar to Virtual-U. Analyzing the LiveBOOKs model we see that this structure has to be recursive (so we can define sections inside sections, for example). As a result of this analysis we produce a kernel that can handle all these requirements. Note that this step is creative, but can be supported by tools [2].

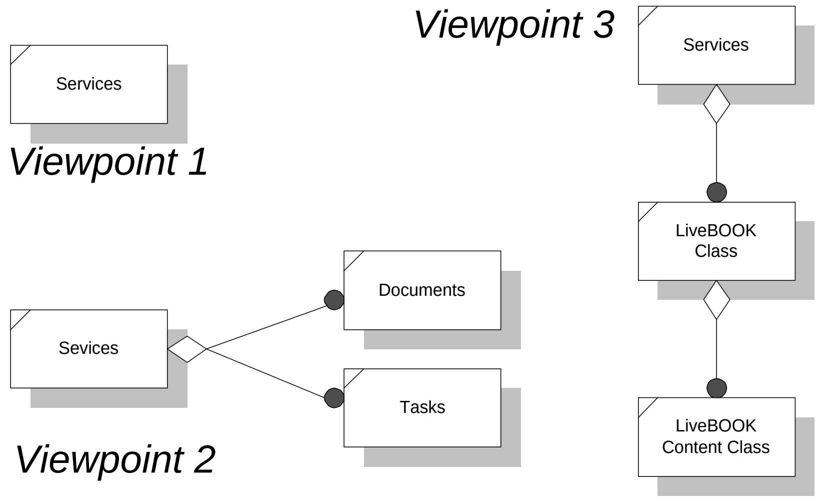


Figure 20. Services and Documents Relationship Viewpoints

We also used the DSL approach to help in the kernel definition. We have defined each environment's class structure through a set of PROLOG-like rules. After analyzing all the environments we are able to define a DSL for WBE that is presented in the next section. As an example, we show here the rules defining the services for the AulaNet and LiveBOOKs environments.

- AulaNet:

```
service("Agenda", _)
service("Course News", _)
service("Other Instructors", _)
service("Test", _)
service("Test Result", _)
service("Project", _)
service("Project Resut", _)
service("Exercice", _)
service("Exercice Result", _)
service("Lesson Plan", _)
service("Transparency", "FileUpload(Powerpoint)")
service("Recorded Presentation", "FileUpload(RealVideo)")
service("Lesson Text", "FileUpload(RichTextFormat)")
service("Text Book", "FileUpload(RichTextFormat)")
service("Demonstration", "FileUpload(RealVideo, Image)")
service("Bibliography", _)
service("Interest Group", "News")
service("Discussion Group", "Listserv")
service("Debate", "Chat")
service("Tutorial about Internet", "Tutorial about Internet")
service("Student Home-Page", _)
service("Search", "Search")
service("Contact with Teacher", "Contact with Teacher")
```

- LiveBOOKS

```
service("LiveBOOK", _)
service("Margin Notes", _)
service("Self Test", _)
service("Eletronic Library", _)
service("Contact with Student", "Contact with Student")
service("Contact with Tutor", "Contact with Tutor")
service("Contact with Marker", "Contact with Marker")
```

By analyzing these two sets of clauses we can see that the service definition has to be a hot-spot in the derived framework, since different environments (viewpoints) provide different services and use different external tools.

The structure of the final object-oriented framework is now presented using the OMT notation. Figures 21, 22, and 23 illustrate the main classes in the WBE framework. The structure is very similar to AulaNet's structure. The main differences are that the framework supports the concept of documents, which are not handled by AulaNet, and that the types of actors (Figure 22) are now a hot-spot, and are not restricted to student, teacher, and author. Each one of these types can also be assigned to a type level, which allows us to categorize the authors. New types of actors and type levels can be dynamically added to the framework. The concept of tasks, presented in LearningSpace and Virtual-U are not directly supported by the framework because it can be implemented as a service, in a more elegant way. Documents are represented by the framework as a recursive structure (Figure 23), making possible the development of recursive services, such as the book structure in the LiveBOOKs environment. The framework is implemented in a way that more resources may be added dynamically, as well as the services and external tools used to implement them.

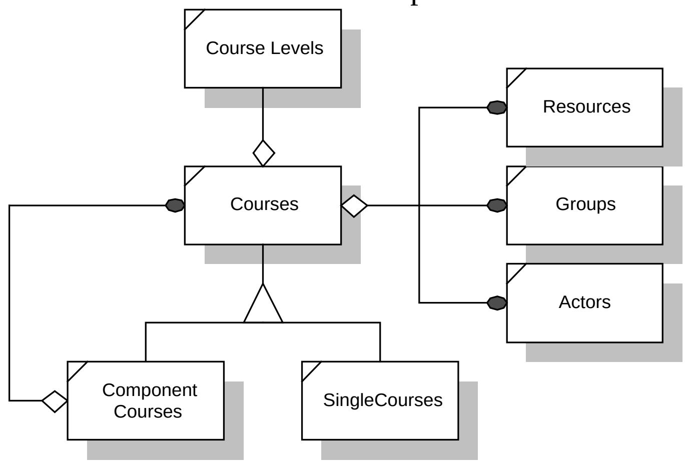


Figure 21. Courses OMT Class Diagram

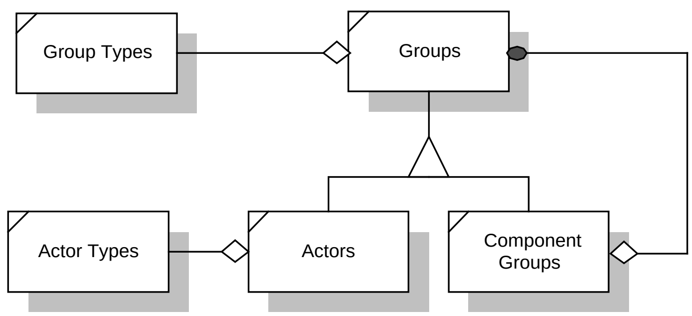


Figure 22. Actors and Groups Diagrams

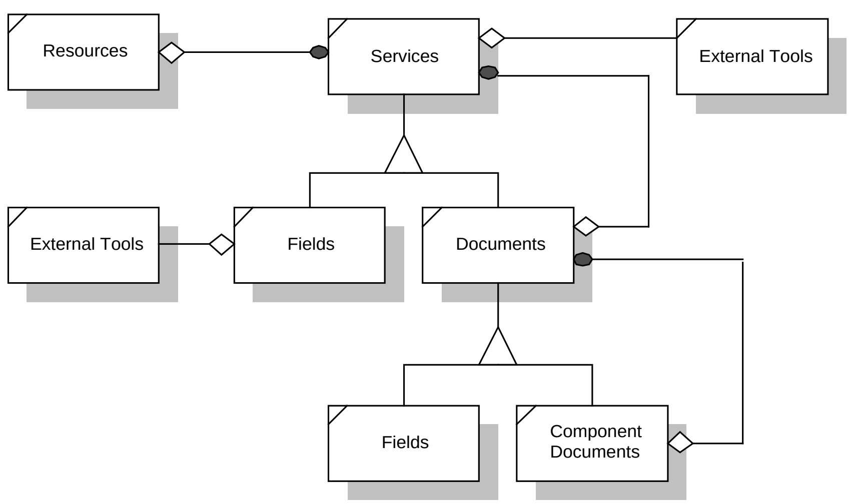


Figure 23. Resources and Services

The framework also supports the fact that each institution and department may have its own set of resources and services. This feature guarantees that the frameworks completely match the requirements of different institutions. This structure is illustrated in Figure 24.

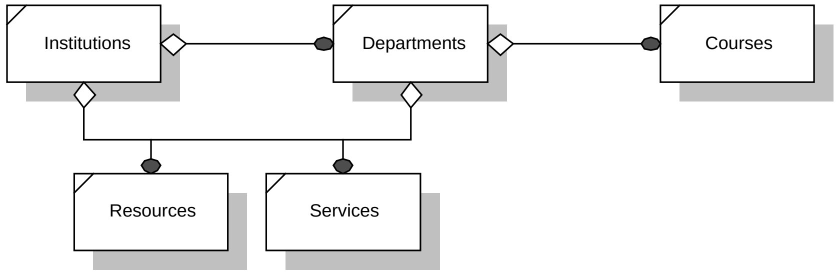


Figure 24. Institutions and Departments Structure

Each course may also configure the actors (and groups) types that are responsible for performing each of its services. This structure for actors is shown in Figure 25.

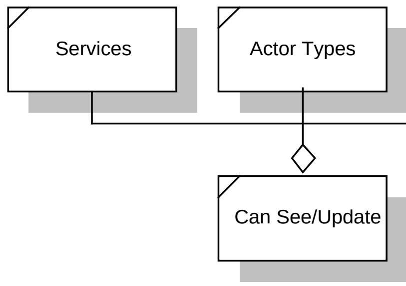


Figure 25. Actors performing services

The visual representation (Interface) for each one of the framework's entities is also a framework hot-spot and the structure is shown in Figure 26. As in the analyzed WBE environments the authoring and learning sites are also hot-spots.

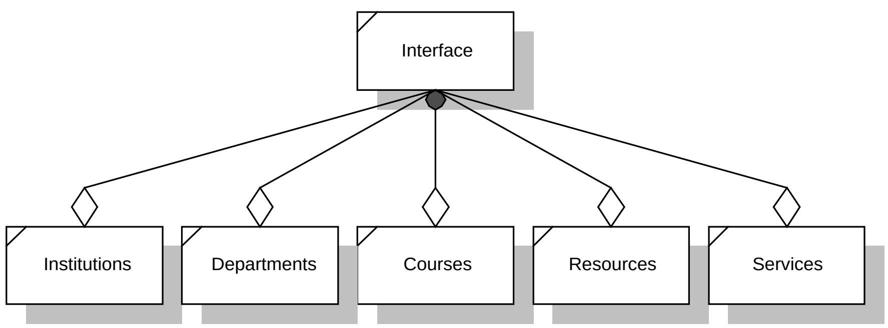


Figure 26. Interface Hot-spot

4.2 Domain-Specific Language

Based on the framework's class structure we have developed a domain-specific language[1] (DSL) for framework instantiation and evolution, which we called WBELang. The basic idea is that the DSL can be used to customize the framework for the generation of specific applications. The DSL must deal only with entities of the domain, being more expressive than wide-spectrum (Turing complete) languages for the specification of applications in that domain.

The WBE domain is a specialization of the domain of web-based applications, which combines the WWW navigational structure with the data access methods that

[1] The definition as a "language" is informal, since it is really a set of rules and facts.

manipulate a server side database. In general these applications are implemented via CGI scripts. In WBE applications the navigational aspects can be implemented in the usual way, and in the actual implementation of our framework they will be handled by the Spider Internet architecture [17].

The database access methods are usually implemented by a relational database and SQL queries. WBELang provides a higher level language than SQL for WBE data manipulation. Using this language the user does not need to know SQL to construct WBE applications. As an example we show the navigational structure of the AulaNet authoring site (Figure 27). Next we describe how this application is developed using the DSL presented here with CGILua [11].

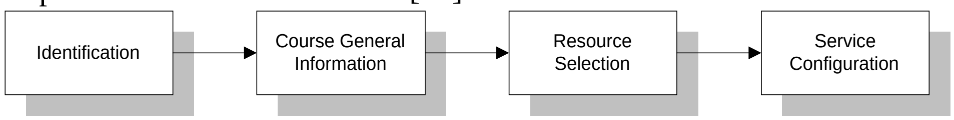


Figure 27. AulaNet Authoring Site: Global View

In the identification step the author has to provide an e-mail 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. 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 WEBLang embedded code) generated by the Spider WBE meta-environment for the resource selection step is shown below.

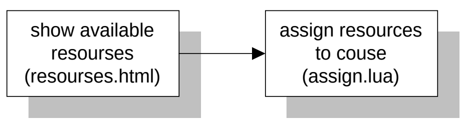


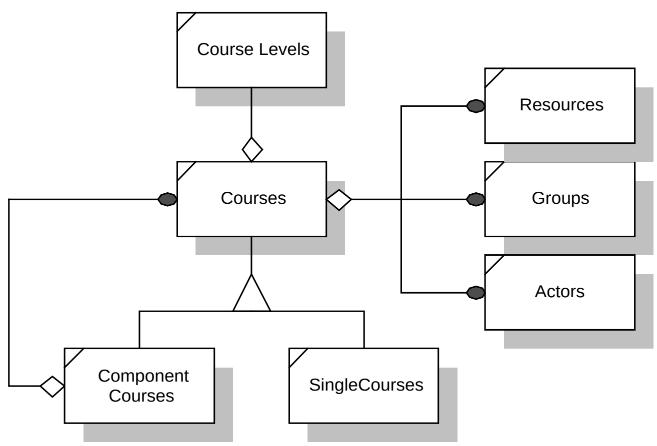
Figure 28. Resource Selection: Detailed Structure

```
****************************************
*             resources.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 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 = weblang("resource(_, _, 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">
</FORM>
```

```
</BODY>
</HTML>

****************************************
*               assign.lua             *
****************************************
table = weblang("resource(_, _, 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
```

The language was developed by analyzing the object models of all the WBE environments and defining PROLOG-like rules to specify their entities. The unification of these sets of rules generated the framework's kernel and the WBELang's syntax. Figure 29 illustrates the WBELang code that represents the framework's document class structure, as an example. The complete language is presented in Appendix A.

Users can instantiate the framework by "programming" in WBELang. In fact, as we have developed a black-box framework, its instantiation and evolution can be made either by plugging components into the hot-spots or via parameterization. Once the framework's DSL is declarative and well defined, the framework can be instantiated with parameters. This leads to the possibility of defining wizards that allow the language utilization via parameter definition, and as a consequence simplification of the framework.




```
course(name, authorName, code, description, syllabus,
institutionName, departmentName, idiomName, couseLevelName) <--
        actor(authorName, _, _, _, _, _, _)
        institution(institutionName, _)
        department(departmentName, _)
        idiom(idiomName)
        courseLevel(couseLevelName, _)

isMadeOf(courseName1, courseName2) <--
        course(courseName1, _, _, _, _, _, _, _, _)
        course(courseName2, _, _, _, _, _, _, _, _)

has(courseName, resourceName) <--
        course(courseName, _, _, _, _, _, _, _, _)
        resource(resourceName, _, _, _)

actorInCourse(actorName, courseName, actorTypeName,
actorLevelName) <--
        actor(actorName, _, _, _, _, _, _)
        course(courseName, _, _, _, _, _, _, _, _)
        actorTypeLevel(actorLevelName, actorTypeName)
```

Figure 29. WBELang and Class Structure for Courses

5. CONCLUSION

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

The viewpoint analysis presented here and the associated conceptual model are the theoretical basis for the production of such an environment. This environment is now being developed and tested [6].

The idea of using existing environments as viewpoints is important for understanding the domain. The analysis process presented here also helps us to validate our viewpoint-based framework development method [2].

APPENDIX A – WBELang

```
*******************************************************
*              Web-Based Education - WBELang          *
*******************************************************

fieldType("boolean")
fieldType("integer")
fieldType("real")
fieldType("string")
fieldType("memo")
fieldType("date")
fieldType("file")
fieldType("blob")
fieldType("imageMap")

fieldType("actor")
fieldType("group")
fieldType("course")
fieldType("service")
fieldType("document")
fieldType(serviceName) <--
       service(serviceName, _)
fieldType(documentName) <--
       document(documentName)

resourceType(name, institutionName, departmentName) <--
       institution(institutionName, _)
       department(departmentName, _)

resource(name, resourceTypeName, institutionName, departmentName) <--
       resourceType(resourceTypeName, institutionName, departmentName)

service(name, externalToolName) <--
       externalTool(externalToolName, _)

document(name)

isPartOf(serviceName, documentName) <--
       service(seviceName, _)
       document(documentName)

isPartOf(document1, document2) <--
       document(document1)
       document(document2)
       document1 != document2

serviceField(serviceName, fieldName, fieldTypeName, isNecessary, externalToolName) <--
       service(serviceName, _)
       fieldType(fieldTypeName)
       isNecessary == "yes" || isNecessary == "no"
       externalTool(externalToolName, _)

serviceField(compositeFieldName, fieldName, fieldTypeName, isNecessary,
externalToolName) <--
       document(compositeFieldName)
       fieldType(fieldTypeName)
       isNecessary == "yes" || isNecessary == "no"
       externalTool(externalToolName, _)
```

```
externalTool(name, executes)

mapping(resourceName, serviceName) <--
        resource(resourceName, _, _, _)
        sevice(serviceName, _, _)

idiom(name)

institution(name, decription)

department(name, description)

actorType(name, description)

actorTypeLevel(name, actorTypeName) <--
        actorType(actorTypeName, _)

actor(name, personalDescription, institutionName, departmentName, url, email,
password) <--
        institution(institutionName, _)
        department(departmentName, _)

canUpdate(serviceName, courseName, actorTypeName) <--
        service(serviceName, _)
        course(courseName, _, _, _, _, _, _)
        actorType(actorTypeName, _)

canSee(serviceName, courseName, actorTypeName) <--
        service(serviceName, _)
        course(courseName, _, _, _, _, _, _)
        actorType(actorTypeName, _)

canSee(X, Y, Z) <-- canUpdate(X, Y, Z)

groupType(name, description)

group(name, groupDescription, groupTypeName) <--
        groupType(groupTypeName, _)

add(actorName, groupName) <--
        actor(actorName, _, _, _, _, _, _)
        group(groupName, _, _)

add(groupName1, groupName2) <--
        group(groupName1, _, _)
        group(groupName2, _, _)

canUpdate(serviceName, courseName, groupTypeName) <--
        service(serviceName, _)
        course(courseName, _, _, _, _, _, _)
        groupType(groupTypeName, _)

canSee(serviceName, courseName, groupTypeName) <--
        service(serviceName, _)
        course(courseName, _, _, _, _, _, _)
        groupType(groupTypeName, _)

courseLevel(name, description)

course(name, authorName, code, description, syllabus, institutionName, departmentName,
idiomName, couseLevelName) <--
        actor(authorName, _, _, _, _, _, _)
        institution(institutionName, _)
        department(departmentName, _)
        idiom(idiomName)
        courseLevel(couseLevelName, _)

isMadeOf(courseName1, courseName2) <--
        course(courseName1, _, _, _, _, _, _, _, _)
        course(courseName2, _, _, _, _, _, _, _, _)

has(courseName, resourceName) <--
        course(courseName, _, _, _, _, _, _, _, _)
        resource(resourceName, _, _, _)

actorInCourse(actorName, courseName, actorTypeName, actorLevelName) <--
        actor(actorName, _, _, _, _, _, _)
        course(courseName, _, _, _, _, _, _, _, _)
        actorTypeLevel(actorLevelName, actorTypeName)
```

```
serviceInstance(courseName, serviceName, fieldName, value) <--
       course(courseName, _, _, _, _, _, _, _, _)
       serviceField(serviceName, fieldName, _, _, _)

serviceInstance(courseName, documentName, fieldName, value) <--
       course(courseName, _, _, _, _, _, _, _, _)
       serviceField(documentName, fieldName, _, _, _)

interface(resourceTypeName, idiomName, imageFile, description) <--
       resourceType(resourceTypeName, _, _)
       idiom(idiomName)

interface(resourceName, idiomName, imageFile, description) <--
       resource(resourceName, _, _, _)
       idiom(idiomName)

interface(serviceName, idiomName, imageFile, description) <--
       service(serviceName, _)
       idiom(idiomName)

interface(institutionName, imageFile) <--
       institution(institutionName, _)

interface(departmentName, imageFile) <--
       department(departmentName, _)

interface(actorName, imageFile) <--
       actor(actorName, _, _, _, _, _, _)

interface(courseName, imageFile) <--
       course(courseName, _, _, _, _, _, _, _, _)
```

APPENDIX B – WBE Services List

The following table summarizes the services implemented for each one of the analyzed environments. The services are classified here in four categories: communication, administrative, evaluation, and teaching services.

AulaNet	Communication Services	Interest Group Discussion Group Contact with teacher Debate
	Administrative Services	Agenda Course News Other Instructors
	Evaluation Services	Test Test Result Project Project Result Exercise Exercise Result
	Teaching Services	Lesson Plan Transparency Recorded Presentation Lesson Text Text Book Demonstration Bibliography
	General Services	Tutorial about Internet Student Home-Page Search
LiveBOOKs	Communication Services	Contact with Student Contact with Tutor Contact with Marker
	Evaluation Services	Self Test
	Teaching Services	LiveBOOK Margin Notes Electronic Library
Web Course in a Box (WCB)	Communication Services	Contact with teacher Chat
	Administrative Services	Announcements Schedule
	Teaching Services	Class Info Learning Links
	General Services	Student Home-Page Instructor Info WCB Forum

WebCT	Communication Services	Electronic mail News Chat
	Administrative Services	Bulletin-board Grade tool Student progress tracking Course tracking
	Evaluation Services	Timed quizzes
	Teaching Services	Searchable course glossary Student annotation facility Course reference material Shared, interactive white board
	General Services	Student presentation areas Searchable image archive Automatic indexing and searching
LearningSpace	Communication Services	Electronic mail News
	Administrative Services	Schedule
	Evaluation Services	Assignments Tasks
	Teaching Services	(Media Center) Multimedia documents Images, sounds, and video
	General Services	Profiles Assessment Manager
Virtual-U	Communication Services	Electronic mail News Debates
	Administrative Services	Schedule
	Evaluation Services	Assignments Tasks
	Teaching Services	Student-moderated online seminars Project teams Goal-setting Real-time video conferencing 3D real-time messaging using Worlds Chat
	General Services	Statistics

REFERENCES

1. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, “OwlNet: An Object-Oriented Environment for WBE”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998.

2. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, “A Viewpoint-based Design Method: A Case Study in Frameworks for Web-based Education”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998.

3. P. Alencar, D. Cowan, C. J. Lucena, and L.C. Nova, “The Views Relationship in Object-Oriented Analysis and Design”, Technical Report, CS-97-13, University of Waterloo, Ontario, Canada, 1997.

4. J. Bell, et al, “Software design for reliability and reuse: A proof-of-concept demonstration”, In TRI-Ada ’94 Proceedings, ACM, 396-404, November 1994.

5. D. Cowan, “An Object-Oriented Framework for LiveBOOKs”, Technical Report, CS-98, University of Waterloo, Ontario, Canada, 1998.

6. D. Cowan, S. Crespo, M. F. Fontoura, C. J. Lucena, and L. M. Moura, “An Architecture for the Visual Programming Development of Web-Based Education Environments”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998.

7. D. Cowan and C. J. P. Lucena, “Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse”, IEEE Transactions on Software Engineering, March 1995.

8. E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, “Design Patterns, Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.

9. M. Goldeberg and S. Salari, “World Wide Web-Course Tool: An Environment for Building World Wide Web-based Courses”, Computer Networks and ISDN Systems, 28, 1212-1231, 1996.

10. P. Hudak, “Building Domain-Specific Embedded Languages”, Computing Surveys, 28A(4), ACM, 1996.

11. R. Iersalimschy, R. Borges, and A. M. Hester, “CGILua A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages”, SBES’97 (Simpósio Brasileiro de Engenharia de Software), 1997.

12. Internet 2, <http://www.internet2.edu>.

13. ICC, <http://www.les.inf.puc-rio.br/icc> (in Portuguese).

14. R. Johnson, “Frameworks = (Components + Patterns)”, Communications of the ACM, Volume 40, Number 10, October 1997.

15. LearningSpace, <http://www.lotus.com/home.nsf/welcome/learnspace>.

16. C. Lucena, 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, “AulaNet - An Environment for the Development and Maintenance of Courses on the Web”, in ICEE’98, 1998 (to appear).

17. L. M. Moura and C. J. Lucena, “O Ambiente Visual Spider para o Desenvolvimento de Aplicacoes para a Internet”, MCC36/97, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1997.

18. L. M. Moura, C. J. Lucena, and A. v. Staa, “The Spider Environment”, MCC46/97, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1997 (also submitted to Software Practice and Experience).

19. W. Pree, “Design Patterns for Object-Oriented Software Development”, Addison-Wesley, 1995.

20. D. Roberts and R. Johnson, “Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks” in “Pattern Languages of Program Design 3”, Addison-Wesley, 1997.

21. H. A. Schmid, “Systematic Framework Design by Generalization”, Communications of the ACM, Volume 40, Number 10, October 1997.

22. Simon Fraser University, <http://virtual-u.cs.sfu.ca/vuweb/>.

23. Sociedade da Informação, <http://www.les.inf.puc-rio.br/socinfo> (in Portuguese).

24. Transferência de Calor, <http://www.les.inf.puc-rio.br/transcal> (in Portuguese).

25. University of British Columbia, <http://homebrew.cs.ubc.ca/webct/>.

26. Virginia Commonwealth University, <http://views.vcu.edu/wcb/intro/wcbintro.html>.