Using viewpoints to derive object-oriented frameworks: a case study in the web-based education domain

Marcus Fontoura, Sérgio Crespo, Carlos José Lucena
Computer Science Department
Pontifical Catholic University (PUC-Rio)
Rua Marquês de São Vicente, 225
22453-900, Rio de Janeiro, Brazil
{mafe, crespo, lucena}@inf.puc-rio.br

Paulo S. C. Alencar, Donald D. Cowan
Computer Systems Group
University of Waterloo
Waterloo, Ontario
Canada, N2L3G1
{palencar, dcowan}@csg.uwaterloo.ca

ABSTRACT

This paper is an experience report that illustrates the applicability of a viewpoint-based design method for the Web-based education (WBE) domain. The method is a new approach for domain analysis that generates an object-oriented framework from a set of concrete applications. These applications are defined as viewpoints, since they provide different perspectives of the framework domain. Various existent WBE environments have been used as viewpoints in our case study. The design method has been successfully applied for these viewpoints generating the ALADIN framework. The analyzed WBE environments are presented through object-oriented diagrams. The implementation and use of ALADIN is discussed to validate the results of the case study.

KEYWORDS

Viewpoints, domain analysis, object-oriented frameworks, web-based education, framework usability, domain-specific languages.

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 for which the viability 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 (WBE) (Fontoura et. al., 1998), electronic commerce (Ripper, 1999), and computational biology (Andreatta et. al., 1998).

An interesting strategy for constructing new applications for these domains is the development of object-oriented frameworks. An object-oriented framework can reduce the costs of developing applications since it allows designers and implementers to reuse their previous experience on problem solving at design and code levels (Johnson, 1997). Prior research has shown that high levels of software reuse can be achieved through the use of frameworks (Hamu and Fayad, 1998).

A framework models the behavior of a family of applications (Parnas et. al. 1985). Its kernel represents the similarities among the applications and the specific application behavior is provided by the hot-spots (Pree, 1996; Fontoura, 1999).

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 requirement needs is still object of research (Johnson, 1997; Fontoura, 1999). 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 specification and design.

Techniques such as design patterns (Gamma et. al. 1995), meta-object protocols (Kiczales el. al. 1991), refactoring (Johnson and Opdyke, 1993), and class reorganization (Cassais, 1992), have been proposed to support framework design and evolution. However, none of them addresses the problem of helping the framework designer to find and structure framework similarities (frozen spots) and flexible points (hot-spots).

This paper describes a method for structuring object-oriented frameworks based on the analysis of a set of existent applications. These applications are defined as viewpoints of a domain and rules are applied to derive the domain abstractions from viewpoint definitions. The method applicability is illustrated by a large real case study in the web-based education (WBE) domain.

In order to validate the results of the WBE case study the paper describes how the derived framework has been implemented and discusses its usability. Two domain-specific languages (DSLs) (Hudak, 1996) that assist the framework instantiation are described. The DSLs facilitate the instantiation of applications and have been defined from the framework design.

The rest of the paper is organized as follows: Section 2 presents the software engineering concepts that serve as a basis for our work. It describes the viewpoint-based design method and a supporting environment used to systematize the method application. Section 3 illustrates the WBE case study. Section 4 describes the implementation and use of the derived framework to validate the experiment. Section 5 outlines the experiences learned in applying the design method to the WBE domain. Section 6 describes some related work. Finally, section 7 presents our conclusions and outlines our future research directions.

2 FRAMEWORKS AND THE VIEWPOINT-BASED DESIGN METHOD

This section presents the software engineering concepts that serve as a basis of our design method: frameworks, viewpoints, viewpoint unification, and meta-pattern application. Examples are introduced to illustrate the main points. A supporting environment used to systematize the method application is also described.

2.1 Frameworks

A framework is defined as a generic software for a domain (Johnson, 1997). It provides a reusable semi-finished software architecture that allows both single building blocks and the design of subsystems to be reused. It differs from a standard application because some of its parts, which are the hot-spots or flexible points (Pree, 1996), may have a different implementation for each framework instance, and are left incomplete during design.

Examples of successful frameworks include Unidraw (Vlissides, 1990), ET++ (Gamma et. al. 1995), and IBM San Francisco (http://www.software.ibm.com/ad/sanfrancisco/).

Current object-oriented design methods completely neglect the importance of helping the framework designer to structure the system's kernel and hot-spots (Pree, 1996). The viewpoint-based design method addresses this problem.

2.2 Viewpoints

The development of complex software systems involves many agents with different perspectives of the system they are trying to describe. These perspectives, or viewpoints, are usually

partial or incomplete. Software engineers have recognized the need to identify and use this multiplicity of viewpoints when creating software systems (Filkelstein et. al., 1992; Ainsworth, 1994).

In this paper viewpoints will be represented as standard object-oriented design artifacts, which may be developed through any OOADM (object-oriented analysis and methods) notation. This paper will use an UML-like notation (Rumbaugh et. al., 1998) to illustrate the examples.

2.3 A general description of the viewpoint-based design method

In (Roberts and Johnson, 1997) 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. The viewpoint-based design method allows the definition of frameworks through the analysis of concrete applications, which are defined as viewpoints of the framework domain.

For the purposes of this paper a domain is a set of viewpoints, and every domain has an associated. Unification rules are defined as a way of describing how applications, or viewpoints, can be combined to compose a framework. They specify a way of structuring the similarities and flexible points of a set of viewpoints in a given framework architecture, as illustrated in Figure 1.

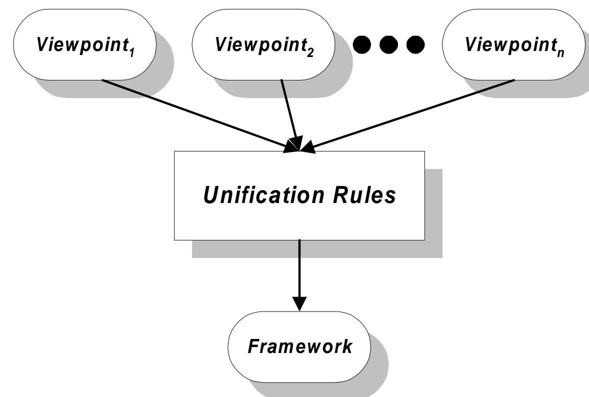

Figure 1. Unification rules

The main idea underlying the unification rules is not to rely on a reverse engineering process that generates a framework from a given set of applications, but to help the framework designer to identify the domain abstractions and better structure the framework architecture. In addition, the use of unification rules highlights the relationship between the kernel (frozen spots) and the hot-spot elements in the framework design.

Once all the relevant viewpoints are defined, the design structure of the framework kernel can be found by analyzing the viewpoints representations and obtaining a resulting design representation that reflects a structure common to all viewpoints. This common structure is the *"unification"* of the viewpoints. This part of the design method is based on the semantic analysis of viewpoint diagrams to discover the common concepts that will compose the design of the framework kernel.

The elements that are not in the kernel are the ones that vary and depend on the use of the framework. These elements define the framework hot-spots (Pree, 1996) that must be adaptable to each application that may be instantiated from the framework. Each hot-spot represents an aspect of the framework that may have a different implementation for each framework instantiation.

Unification rules can not automatically generate the final framework design since the viewpoints generally represent concrete applications, and for this reason, the semantics of how to define the flexible part of the framework is not present in their design. An intermediate artifact, which separates the kernel and hot-spot elements, has to be generated before the generation of the framework design. This intermediate artifact is the “template-hook model”, which is based in template and hook methods (Pree, 1996).

Template methods model the kernel elements which are responsible for invoking the hook methods, which model the hot-spots (Pree, 1996). In the “template-hook model” the relationships between template and a hook classes are defined as “hot-spot relationships”, where the template classes contain the template methods and hook classes contain the hook methods. The generation of the template-hook model may only be partly automated since human interaction may be required to assist the application of the unification rules.

The next step is to define which meta-pattern (Pree, 1996) should be used to model each hot-spot relationship. The application of meta-patterns defines the hot-spot design structure, generating the final framework design. This part of the method has to be assisted by human actors, since the selection of each meta-pattern to use depends on the hot-spot flexibility requirements, and this information is not present in the viewpoints. Once the meta-patterns have been selected their application may be completely automated, transforming the template-hook model into the framework design.

Figure 2 illustrates the whole process in more detail. The next two subsections detail the unification of viewpoints, which generates the template-hook model, and the application of meta-patterns, which generates the framework design.

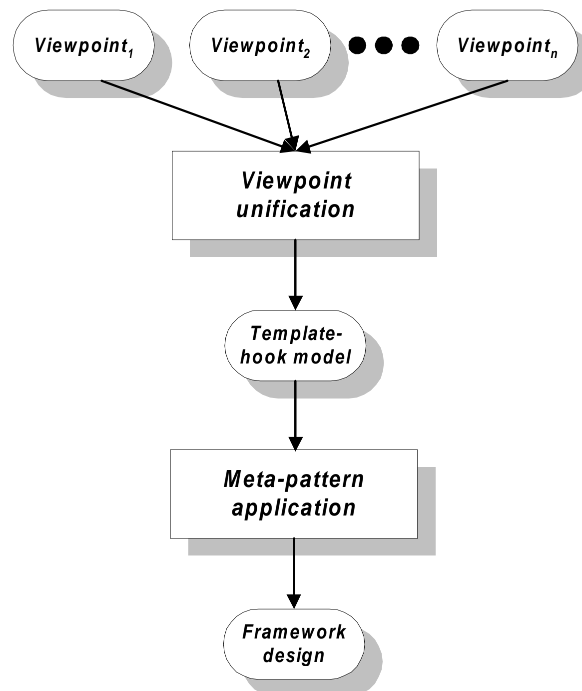

Figure 2. General view of the process

2.4 Viewpoint unification

A more detailed description of the viewpoint unification process is now presented. This process imposes some pre-conditions on the viewpoints that are going to be unified:

1. Same concepts must be represented by the same names: the unification rules will use names to define the semantics of the involved concepts. Thus, if two different viewpoints had names *Report* and *Form* to represent the same concept, a refactoring procedure

(Johnson and Opdyke, 1993) would have to be used to rename one of them. The renaming refactorings can be applied to classes, attributes, and methods and their application are behavior preserving as shown in (Johnson and Opdyke, 1993), meaning that their application does not change the viewpoints semantics;

2. Orthogonal concepts: all classes, methods, and attributes in the initial set of viewpoints must model orthogonal concepts. For example, if one viewpoint had a *generateFormat* method that implements the functionality provided by methods *generateReport* and *format* defined in another viewpoint, a refactoring procedure would have to be used to split *generateFormat* into two methods. Always that *method-p* defined in *viewpoint-i* is non-orthogonal to *method-q* defined in *viewpoint-j*, a behavior preserving decomposition can be applied to transform the methods making them orthogonal. The same holds for classes and attributes;
3. Attribute types must be consistent: attributes representing the same concept (and for this reason having the same name in all viewpoints) should also have the same type. That means that if the same attribute is an integer in one application and a float in other, the applications are inconsistent and the unification rules will not deal with these variations;
4. Cyclic hierarchies must be avoided: the unification of viewpoints cannot generate an inheritance cycle. If there is a *viewpoint-i* within a certain class hierarchy such that *class-sub* is a subclass of *class-super* and a *viewpoint-j* in which *class-super* is a subclass of *class-sub* there is an inconsistency and the unification rules cannot be applied. Note that this problem is generally related to the domain model, since there is an inconsistency, for example, in modeling *Employee* as a subclass of *Person,* and *Person* as a subclass of *Employee* in two different viewpoints of the accounting domain.

Normally some cases class restructuring approaches (Casais, 1992; Johnson and Opdyke, 1993) can be applied to handle the inconsistency. The verification of the above preconditions and the application of restructuring transformations to solve the inconsistency have to be supported by human actors.

When the set of viewpoints is consistent unification rules can be applied. The result of the unification process is a template-hook model. The unification process is based on the following rules:

1. Every class that belongs to the set of viewpoints has a corresponding class, with same name, in the template-hook model;
2. If a method has the same signature and implementation in all the viewpoints it appears, it has a corresponding method, with same name, signature, and implementation, in the template-hook model;
3. If a method exists in more than one viewpoint with different signature it has a corresponding hook method in the template-hook model, with same name but undefined signature and implementation;
4. If a method exists in more than one viewpoint with different implementation[1] it has a corresponding hook method in the template-hook model, with same name and signature but undefined implementation;
5. All the methods that use hook methods are defined as template methods. There is always

[1] This check cannot be automatically performed since it is an undecidable procedure, and has to be supported by human actors.

a hot-spot relationship between the class that has a template method and the class that has its correspondent hook method;

6. All the existing relationships in the set of viewpoints that have no corresponding hot-spot relationship are maintained in the in the template-hook model.

Figure 3 shows an example of two consistent viewpoints that are going to be unified.

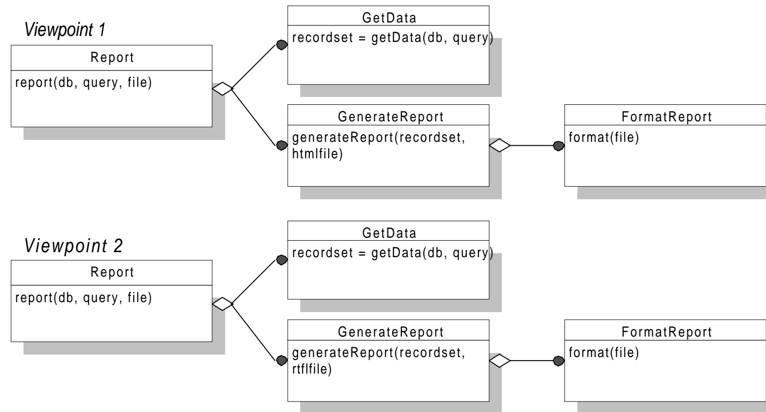

Figure 3. Viewpoint unification example

Viewpoint 1 shows a design diagram of a simple report generator. Its main class is *Report,* which uses the classes *GetData* to access the information required by the report and *GenerateReport,* to generate the final report in an HTML file. Class *GenerateReport* uses class *FormatReport* to configure the layout of the generated HTML file.

Viewpoint 2 follows basically the same structure. Classes *Report* and *GetData* have the same methods, with same signature and implementation. Method *generateReport* (in class *GenerateReport)* has different signatures in the two viewpoints, since one asks for an HTML file while the other asks for an RTF file. In this case rule 3 implies that it is a hook method. Method *format* (in class *FormatReport)* has different implementations in the two viewpoints, since one configures HTML files while the other configures RTF files. Rule 4 defines this method as another hook method. Rule 5 states that all methods that use hook methods are template methods. Thus, in this example, the template methods are *report* and *generateReport*.

The template-hook model presented in Figure 4 is the result of the unification of viewpoints. The dashed arrows represent the hot-spot relationships. Note that the signature of method *generateReport* is UNDEFINED, as established by rule 3, and the implementations of methods *generateReport* and *format* are also UNDEFINED, as established by rules 3 and 4, although not shown in the diagram.

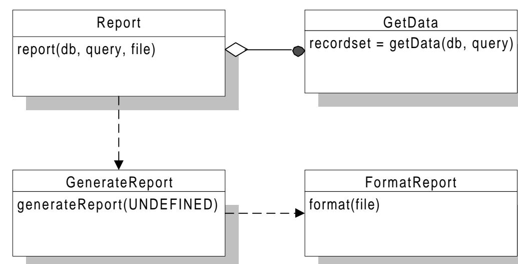

Figure 4. Template-hook model

All undefined signatures and implementations must be defined by the framework user, during instantiation, as will be exemplified in section 4.

2.5 Applying meta-patterns

The framework design is generated from the template-hook model. The framework designer must eliminate all hot-spot relationships from the template-hook model, replacing them by the most appropriate meta-pattern. Hot-spot cards (Pree, 1996) are used to assist this process.

Figure 5 shows the hot-spot card layout, which is a variation of the one presented in (Pree, 1996). There are two flexibility properties shown in the card: adaptation without restart and recursive combination. The combination of these properties is used to select which meta-pattern should be applied.

Table 1 shows the mappings between flexibility properties and meta-patterns. In the unification meta-patterns the template and hook methods belong to the same class and adaptations can be made only through inheritance, which requires the application restart for the changes take effect. In the separation meta-patterns the template and hook methods appear in different classes and adaptations can be made at runtime through the use of object composition.

The recursive combinations of template and hook methods allow the creation of direct object graphs, like the Composite design pattern presented in (Gamma et. al., 1995).

	Adaptation without restart	Recursive Combination	Meta-pattern
1			Unification
2	☑		Separation
3		☑	Recursive Unification
4	☑	☑	Recursive Separation

Table 1. Mappings between flexibility properties and meta-patterns

As an example, let us consider the hot-spot relationships in the template-hook model shown in Figure 4. Suppose that it is necessary the definition of new *generateReport* methods in the system at runtime. Also suppose that the *format* method does not need to be redefined during runtime. Since neither of these relationships requires a recursive combination, the hot-spot cards that represent their flexibility properties are presented in Figure 5.

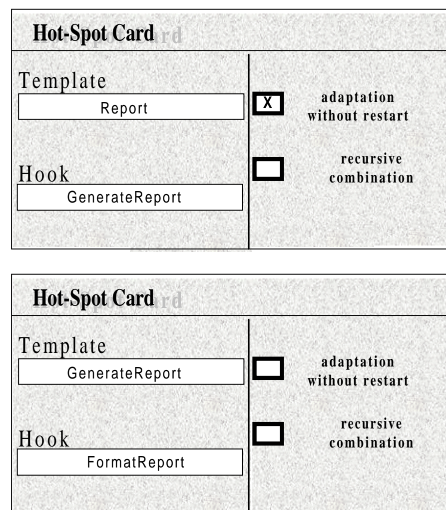

Figure 5. Hot-spot card utilization example

The result diagram after the meta-patterns application is the framework design presented in Figure 6.

Since the meta-pattern used for the hot-spot relationship between classes *GenerateReport* and *FormatReport* unifies the template and hook methods in the same class, class *FormatReport*

is not required in the framework design. A very important property of the method is the control of the design complexity, leading to simple and readable designs. In this example, the generated framework design has less classes than each of the original viewpoints.

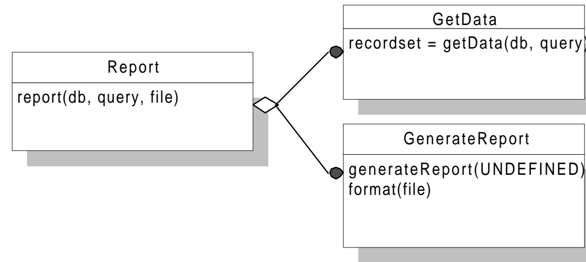

Figure 6. Framework design

2.6 Supporting environment

A supporting environment has been defined to assist the viewpoint-based design method. Since refactoring rules can handle viewpoint inconsistencies, the environment allows the use of refactoring procedures before performing the viewpoint unification. The environment has been defined as a meta-environment to allow for the configuration of new unification rules, new refactoring rules, and the object-oriented model used to describe the viewpoints. The meta-artifacts are described by scripting languages used to configure the environment. This is necessary since the actual unification rules may evolve with the discovery of new patterns, design concepts, or even with the improvement of current object-oriented design languages.

Figure 7 illustrates the meta-environment architecture, highlighting its inputs (meta information and viewpoints) and outputs (refactored viewpoints and frameworks). Its current implementation runs on a WWW browser and has been developed using CGI scripts and relational databases for storing the viewpoints, frameworks, and unification and refactoring rules. The environment currently does not support graphical representation of applications and frameworks, and demands text-based definition of classes and relationships, as shown in Figure 8. The application of unification and refactoring rules must be always assisted by the framework designer, who has to interact with the environment in several steps of the process. Examples of tasks that must be assisted by the framework designer are informing whether a method implementation varies or not and the selection of the most adequate meta-pattern to model each hot-spot relationship.

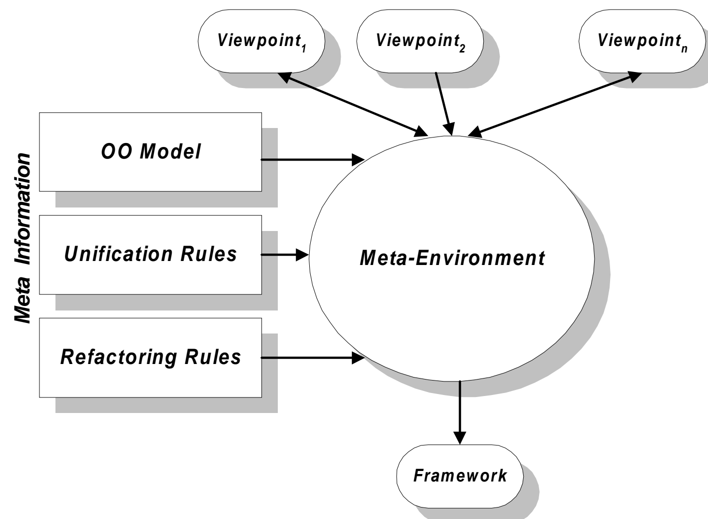

Figure 7. Supporting meta-environment architecture

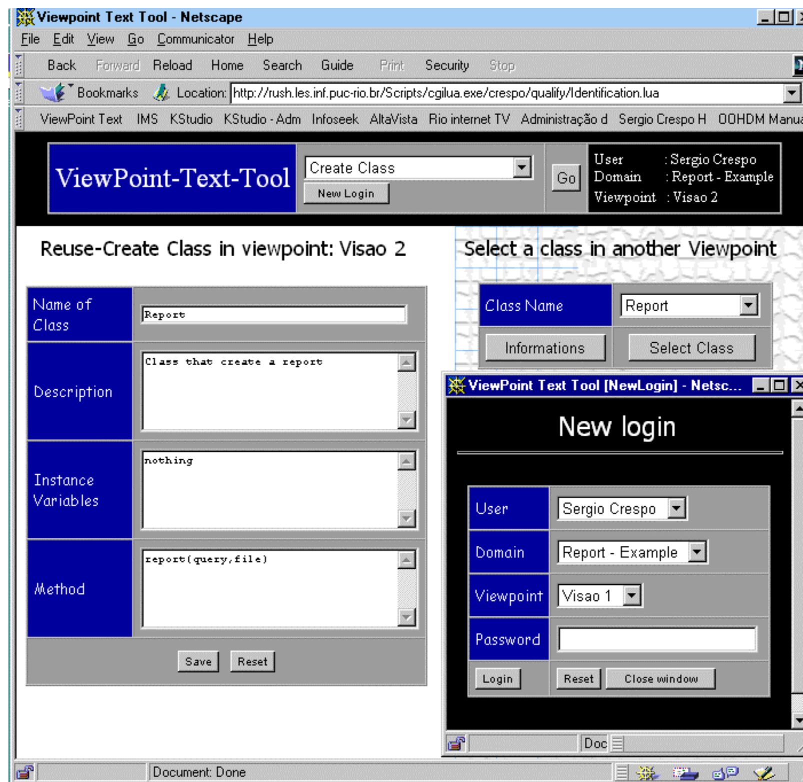

Figure 8. Current implementation

3 THE WBE DOMAIN CASE STUDY

This section presents the models of the analyzed WBE environments and describes the application of the viewpoint-based design method to generate the ALADIN framework. Six existent WBE environments have been used in the case study. The selection of these environments was based on three criteria: (i) their popularity and importance; (ii) the availability of documentation; and (iii) their underlying concepts, so that environments that do not model new concepts have been discarded.

Although there are several others environments available, the six selected environments are very representative and cover the great majority of domain concepts so far discussed in literature (Papert, 1996). Each of these models was considered as a different viewpoint the WBE domain, and the viewpoint unification was used to define the framework design.

Two aspects must be considered when analyzing the following models. First, except for AulaNet and LiveBOOKs, we do not know the exact object-model of the analyzed environments. The models presented here have been specified by the use of these environments. Second, when the models are similar we just refer to the figure that describes it in order to avoid presenting similar models twice.

Since the objective of our comparison of WBE environments is the definition of a framework design, we were not interested in a feature by feature analysis of the environments. Only the core entities of the WBE domain have been analyzed. A useful concept adopted throughout all this analysis was the concept of services. A service has been defined as an atomic functionality provided by the environment. Examples of services are discussions groups, course news, quizzes, and bulletin boards.

3.1 AulaNet

AulaNet (Crespo et. al., 1998) is a WBE environment developed at the PUC-Rio Software Engineering Laboratory. AulaNet allows several institutions to use the environment simultaneously. Each institution may have several departments. The courses are related to institu-

tions and each course has assigned actors. A course consists of a selection of services. This design structure is presented in Figure 9.

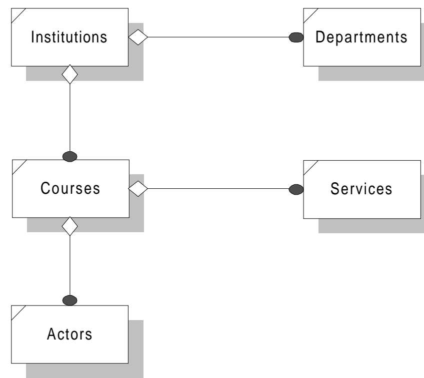

Figure 9. AulaNet OMT class diagram

The AulaNet 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 authors use the authoring site to create and maintain the courses. The class structure that implements both sites is shown in Figure 10, where class *Idioms* represents the support to multiple languages (English, Portuguese, and so on).

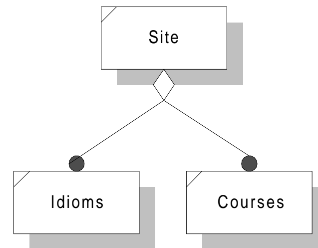

Figure 10. AulaNet site class structure

3.2 LiveBOOKs

The LiveBOOKs (Cowan, 1998) distributed learning/authoring environment is a computer-based teaching/learning and authoring system that supports learning and authoring activities. LiveBOOK class structure is very similar to AulaNet's, except of two main differences: it is does not support the definition of many institutions and departments and the actors types are not restricted to student and author, as show in Figure 11. LiveBOOKs allows new types to be defined as subclasses of *ActorTypes*.

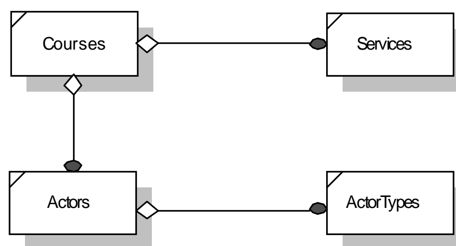

Figure 11. LiveBOOKs class diagram

3.3 Web Course in a Box

Web Course in a Box (WCB) (http://views.vcu.edu/wcb/intro/wcbintro.hml) is a course creation and management tool for web-assisted delivery of instruction. The main difference of this environment and the other two previously presented is that in WCB the final user can modify the visual representation *(Interface)* for each one of its entities as shown in Figure 12 and Figure 13.

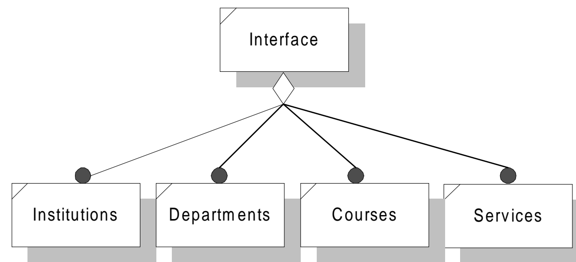

Figure 12. WCB interface class structure

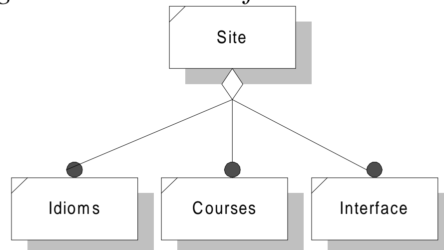

Figure 13. WCB site class structure

3.4 Web-CT

Web-CT (http://homebrew.cs.ubc.ca/webct/) is a tool that facilitates the creation of sophisticated WBE environments. Web-CT class structure can be seen as an extension of the one present for LiveBOOKs. The new concept is that each service can be of two different types: internal, which is implemented by the environment, and external, which is any WWW application not implemented by the environment but available elsewhere in the Internet. Examples of external services are chat applications, CU-SeeMe, e-mail, and list servers. This design structure is presented in Figure 14.

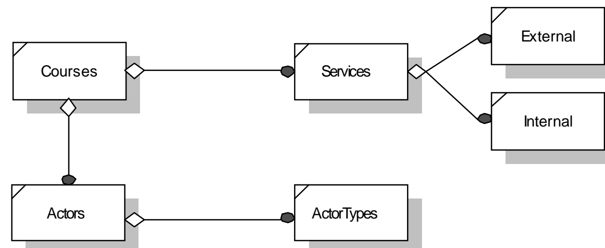

Figure 14. Web-CT class diagram

3.5 LearningSpace and Virtual-U

These two environments are put together here because they have similar class structures. Lotus Education and IBM are responsible for the research and development of Lotus LearningSpace (http://www.lotus.com/home.nsf/welcome/learnspace), an educational technology with supporting services for distance education.

The LearningSpace and Virtual-U (http://virtual-u.cs.sfu.ca/vuweb/) class structures are essentially the same as the one presented in Figure 14. However, they additionally introduce the concepts of documents and tasks (Figure 15). The services are based on documents, and each document may have various tasks assigned to it. In addition, LearningSpace allows for documents and tasks to be classified in categories (Figure 16).

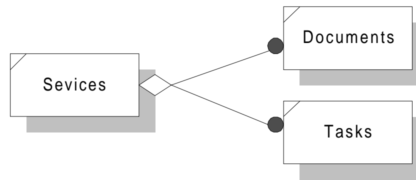

Figure 15. LearningSpace and Virtual-U: services class structure

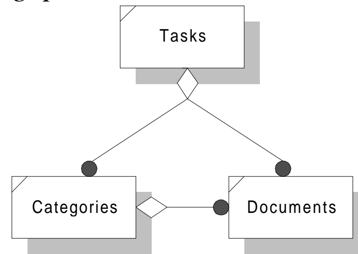

Figure 16. LearningSpace: tasks, categories, and documents

3.6 The viewpoint unification

We have tried to capture the core functionality of the analyzed WBE environments to define an object-oriented framework, called ALADIN (Fontoura et. al., 1998). The viewpoint-based design method has been applied to define the framework kernel and hot-spot structures.

Figure 17 illustrates (in an abstract way) how each one of the analyzed environments has been used as a viewpoint of the final framework. The basic idea was to identify a framework that could provide all the functionality required for generating, at least, the six analyzed environments.

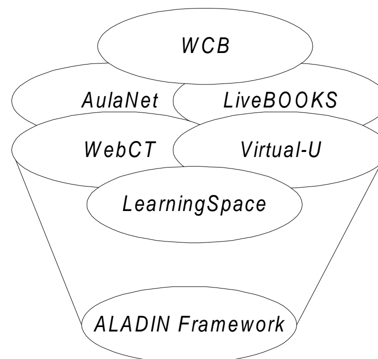

Figure 17. Viewpoints unification (abstractly)

Before applying the unification rules we must verify if viewpoints are consistent. Classes *Tasks, Documents,* and *Categories* (in LearningSpace and Virtual-U) are non-orthogonal to the definition of *Services* in the other viewpoints. It is possible to model *Tasks, Documents,* and *Categories* as *Services* in LiveBOOKS or in AulaNet. To solve this inconsistency these classes have been discarded the template-hook model. Note that this decision is creative and depend on understanding the domain concepts. After this modification the viewpoints have become consistent and could be unified.

Classes *Institutions, Departments, Courses, ActorTypes, Services,* and *Documents* are responsible for accessing their correspondent information in a database system. To provide this functionality access methods *(get/set)* are present in each of these classes. However, the attributes that define these entities vary in each analyzed environment, and therefore all the access methods have different signature and implementation in each viewpoint. Unification rule number 3 may be applied for these methods, generating the template-hook model shown in Figure 18.

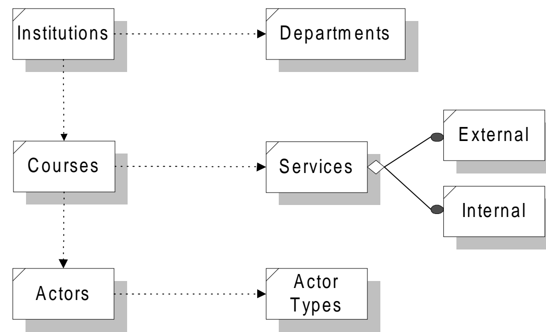

Figure 18. ALADIN template-hook model

Classes *Interface* and *Site* define the visual representation and navigational structure of the WBE applications, respectively. *Interface* methods depend on the layout structure (usually defined by a graphical designer), which may vary for each viewpoint. *Site* methods are responsible for generating the output HTML files, and may also vary from one viewpoint to another. Therefore, unification rule number 4 may be applied for both *Interface* and *Site.* Figure 19 and Figure 20 shows the generated template-hook models.

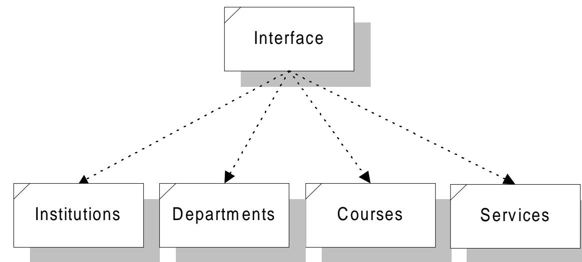

Figure 19. ALADIN interface template-hook model

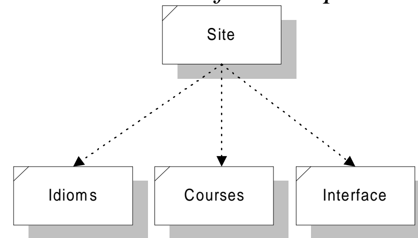

Figure 20. ALADIN site template-hook model

Since adaptation during runtime is an important feature in the WBE domain and recursive combination is not required by any of the hot-spots, the separation meta-pattern was selected to implement all the hot-spot relationships in the template-hook model. The structure of the final framework design is shown in Figure 21, Figure 22, and Figure 23. The UNDEFINED signatures and implementations, elided in the figures, will be completed only during framework instantiation since they may vary for each instantiated application.

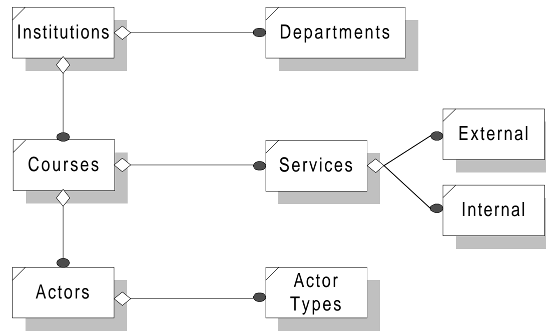

Figure 21. ALADIN class diagram

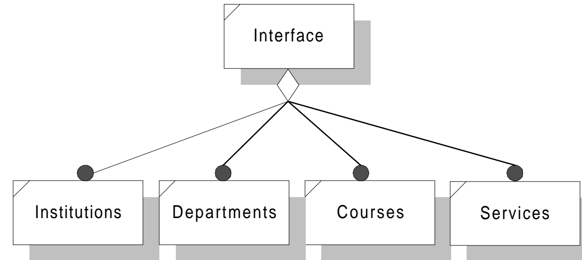

Figure 22. ALADIN interface class structure

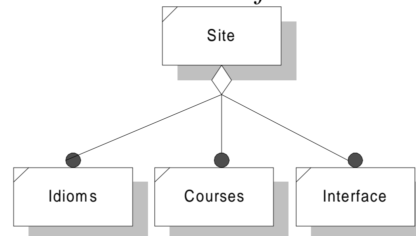

Figure 23. ALADIN site class structure

This section has shown how the design method has been successfully applied to a large case study in the WBE domain. Next section describes the implementation and use of the ALADIN framework.

4 VALIDATING THE GENERATED FRAMEWORK

To verify quality of the frameworks generated from the design method a running version ALADIN has been implemented and tested in the creation of WBE applications.

ALADIN has been implemented in CGILua (Iersalimschy et. al., 1997) following the framework model design structure generated from the viewpoints unification. Two domain specific languages (DSLs) (Hudak, 1996) have been defined to assist its instantiation:

1. Educational Language: used in the definition of the educational components (courses, actors, services, institutions and departments);
2. Navigational Language: used in the definition of the language (e.g. English, French), interface (e.g. background images, buttons), and navigational structure.

The DSLs constructs have been defined form the ALADIN design. Each framework hot-spot has an associated DSL construct. Programs written in these two DSLs are used to complete the missing hot-spot information, marked as UNDEFINED in the framework design.

We estimate that the use of ALADIN increases the productivity by a factor of three. A WBE system that would take six months to be developed may be instantiated from ALADIN in approximately two months, for example. This conclusion was based on several experiments developed by our research group, including the redesign of AulaNet.

The following subsections describe the two DSLs and show an instantiation example.

4.1 Educational Language

Educational Language programs must be transformed to complete the definition of the ALADIN hot-spots. Two files are generated from each Educational program: Database definition and Access methods. The first is used for defining the database structure that will be used by the WBE application being instantiated. The second provides the *get/set* methods for accessing the database. Figure 24 illustrates this architecture.

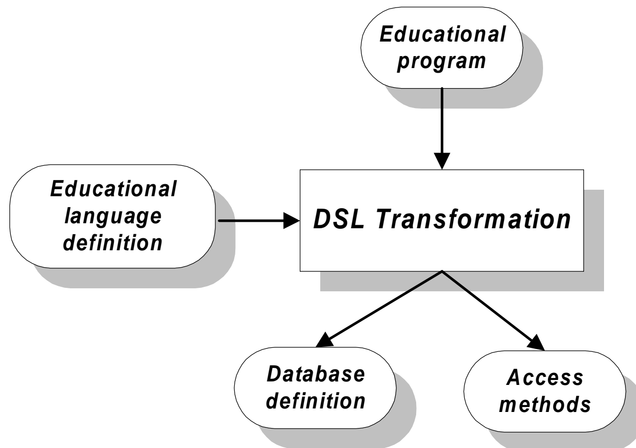

Figure 24. Educational language architecture

The following program code is written in 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 ];
```

Each language construct is used to complete the missing information one hot-spot. The definition of the course attributes in the operator *Course* is an example.

The execution of the Database definition code generates the database that will be used by the WBE application being instantiated. The current implementation of ALADIN generates Microsoft Access databases. One database is generated for each instantiated application.

The following Lua methods (Iersalimschy et. al., 1996), which have been generated from the previously shown Educational program, are part of the Access methods used to access course definitions. Similar access methods are generated for the others hot-spots. These methods encapsulate the SQL commands, allowing ALADIN users to generate WBE environments

without having any knowledge in manipulating databases.

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

4.2 Navigational Language

ALADIN allows the generation of one WWW site for each type of actor defined in the system. Normally two sites are always present in the existent WBE environments: a learning site and an authoring site. However, ALADIN allows other actor types to have their own 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.

The Navigational Language code is transformed into HTML and Lua files that will define the application interface and navigational structure. These files use Access methods generated from the Educational program access all the required information.

An example of Navigational Language program 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. 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 had been Portuguese.

The same is valid for the hypertext links that define the application navigational structure. In the above example, the tag *<ALADIN-LINK> previous </ALADIN-LINK>* in 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 texts, images, and navigational links in a unique file, providing more readability and flexibility. Since the site interface and navigational structure are defined separate form the HTML templates, they may completely redefined without impacting in the rest of the system.

In order to instantiate ALADIN the application developer has to define an Educational program, a Navigational program, and all the required HTML templates.

4.3 ALADIN instantiation example

This section describes how the AulaNet authoring site (http://www.les.inf.puc-rio.br/aulanet) has been redesigned using the ALADIN framework. The general structure of the authoring site is shown in Figure 25.

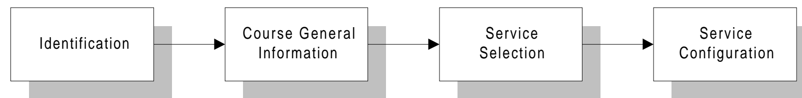

Figure 25. AulaNet authoring site: global view

In the identification step the author has to provide a user-id 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[1]. In the service selection phase the author has to inform the system what services will be used for the course being developed. Finally, in the service configuration phase, each selected service must be configured. Once all these steps have been completed the course is ready and its learning site can be generated. The detailed structure for the service selection step is shown in Figure 26, and HTML code (with embedded CGILua) generated by the ALADIN framework is presented next.

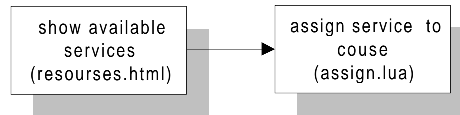

Figure 26. Resource selection: detailed structure

```
****************************************
*             services.html            *
****************************************
<HTML><HEAD>
<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">
<!--$$
-- Select all the services available to the
-- current institution. The result of this
-- selection is stored in a Lua table.
table = getService(currentInstitution)
while (table ~= nil) do
  write('<td valign=middle>')
  write('<INPUT TYPE="Checkbox"
       NAME="'..table[0]..'" VALUE="'..table[0]..'">')
  write('</td>')
  moveNext(table)
end
$$-->
<INPUT TYPE= "Submit" VALUE= "Next">
</FORM></BODY></HTML>

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

The generated HTML and Lua files are simple and easy to be understood since they use Access methods high-level constructs instead of low-level database operations.

5 ANALYZING THE DESIGN METHOD

The ALADIN framework was the first large case study developed using the viewpoint-based

---

[1] The fields to be completed in this step are defined by the *Course* operator, in the Educational program.

design method. The unification transformations have been applied without tool support since the environment described in section 2.6 was not available at the time.

The application of the unification rules was very straightforward because the viewpoints had similar design structures. However, the verification of viewpoint inconsistencies is a very hard task that can only be performed by domain experts. In this case study the verification that *Services* and *Tasks, Documents* and *Categories* were non-orthogonal classes has only been possible because we were familiar with the semantics of these concepts.

The ALADIN implementation and the definition of the DSLs to assist its instantiation process were completely based on the generated design model. ALADIN has been successfully used in the development of several web-based applications, simplifying the construction these application and reducing development costs (Fontoura et. al., 1998). It has not been used outside our research group since its main purpose is to allow for experimentation with educational environments developed at our Lab.

Several other small case studies in other domains have been developed at our research group, generating good results. However, in order to further evaluate the merits and the limitations of the method new large case studies need to be developed. We are now evaluating the applicability of the approach for the electronic commerce domain. This project is being developed with the support of the viewpoint unification tool described in this paper.

6 RELATED WORK

The model for framework development based on viewpoints proposed in (Alencar et. al. 1999) was used as our first approach for framework design, and the current version has been refined through the development of several case studies.

Currently there are very few framework design methods. A pattern-based description of some accepted approaches underlying framework design can be found in (Roberts and Johnson, 1997). However, existing methods do not address the problem of identifying the possible relationships between kernel and hot-spot elements.

Some work in systematization of the application of patterns to implement framework hot-spots can be fund in (Schmid, 1997). Other interesting aspects regarding framework design such as framework integration, version control and over-featuring can be found in (Codenie et. al., 1997).

7 CONCLUSIONS AND FUTURE WORK

This paper shows how unification rules can be used to generate frameworks from an initial set of viewpoints. We believe that this approach leads to a more explicit definition of the collaboration between the framework elements and helps the designer to better implement the framework hot-spots through the use of meta-patterns. Currently comparable methods are almost nonexistent or in their infancy. The analysis process presented here is a large cased study that helped us to validate our viewpoint-based design method.

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, with little help from software engineers.

The viewpoint analysis presented here and the associated conceptual model are the theoretical basis for the production of such a framework. The ALADIN framework is completely developed and has been tested in the generation of new WBE environments (Fontoura et. al., 1998).

A new version of the method supporting environment that provides cooperative work capabilities and graphical representation is now being developed in Java. This tool will be used to experiment with different objet-oriented models and evaluate the impact of these models to framework design. More concretely, the tool will provide an uniform way of validating how other design techniques can enhance framework design, since unification and refactoring rules that can profit from these models will be compared through the environment.

The formalization of unification rules and a systematic approach for generating DSLs from the framework design can be found in (Fontoura, 1999).

8 REFERENCES

Alencar, P. Cowan, D. Nelson, T. Fontoura, F. and Lucena, C., Viewpoints and Frameworks in Component-Based Design, in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John-Wiley, (1999).

Ainsworth, M., Cruickshank, A. H., Groves, L. J., and Wallis, P. J. L., Viewpoint specification and Z, *Information and Software Technology*, 36(1), 43-51 (1994).

Andreatta, A. Carvalho, S. and Ribeiro, C., An Object-Oriented Framework for Local Search Heuristics, 26th TOOLS, *IEEE Press*, 33-45 (1998).

Casais, E., An incremental class reorganization approach, ECOOP'92 Proceedings, *Lecture Notes in Computer Science*, 615, 114-132 (1992).

Codenie, W. Hondt, K. Steyaert, P. and Vercammen, A., From Custom Applications to Domain-Specific Frameworks, *Communications of the ACM*, 40(10), 71-77, (1997).

Cordy, J. and Carmichael, I., The TXL Programming Language Syntax and Informal Semantics, Technical Report, Queen's University at Kingston, Canada, 1993.

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

Crespo, S., Fontoura, M., and Lucena, C., AulaNet: An Object-Oriented Environment for Web-based Education, *International Conference of the Learning Sciences 1998*, 304-306 (1998).

Filkelstein, A., Kramer, J., Nuseibeh, B., Filkelstein, L., and Goedicke,M., Viewpoints: A Framework for Integrating Multiple Perspectives in System Development, *International Journal of Software Engineering and Knowledge Engineering,* 2(1), 31-58 (1993).

Fontoura, M., A Systematic Approach for Framework Development, Ph.D. Thesis, Departamento de Informática, PUC-Rio, 1999.

Fontoura, M., L. Moura, Crespo, S., and Lucena, C., ALADIN: An Architecture for Learningware Applications Design and Instantiation, MCC35/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 .

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

Hamu, D. and Fayad, M., Achieving Bottom-Line Improvements with Enterprise Frame-

works, *Communications of ACM*, 41(8), 110-113, (1998).

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

Iersalimschy, R., Borges, R., and Hester, A. M., CGILua A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages, SBES'97 – Brazilian Symposium on Software Engineering, (1997).

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

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

Johnson, R. and Opdyke, W. F., Refactoring and aggregation, Object Technologies for Advanced Software, First JSSST International Symposium, *Lecture Notes in Computer Science*, 742, 264-278 (1993).

Kiczales, G. des Rivieres, J. and Bobrow D., *The Art of Meta-object Protocol*, MIT Press, 1991.

Papert, S., *The Connected Family*, Longstreet Press, 1996.

Parnas, D. Clements, P. and Weiss, D., The Modular Structure of Complex Systems, *IEEE Transactions on Software Engineering*, SE-11, 259-266, (1985).

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

Ripper, P., V-Market: A Framework for Agent Mediated E-Commerce Systems based on Virtual Marketplaces, M.Sc. Dissertation, Departamento de Informática, PUC-Rio, 1999.

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).

Rumbaugh, J. Jacobson, I. and Booch, G., *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1998.

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

Vlissides, J., Generalized Graphical Object Editing, Ph.D. Thesis, Department of Electrical Engineering, Stanford University, 1990.

The authors:

Marcus Fontoura, is a researcher at the Software Engineering Lab, Computer Science Department, PUC-Rio, Brazil, and a post-doctoral researcher at the Computer Science Department, Princeton University, U.S.A.

Sergio Crespo is a Ph.D. candidate at the Computer Science Department, PUC-Rio, Brazil.

Carlos José Lucena, is a full professor at the Computer Science Department, PUC-Rio, and the director of the Software Engineering Lab at PUC-Rio, Brazil.

Paulo S. C. Alencar, is an associate research professor at the Computer Systems Group, University of Waterloo, Canada.

Donald D. Cowan, is a professor emeritus at the Computer Science Department, University of Waterloo, and the director of the Computer Systems Group, Canada.