Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 233 (2009) 127–142

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Tool Support for Pattern Selection and Use

Bahman Zamani [1],[2] Greg Butler [3] Sahar Kayhani [4]

Department of Computer Science and Software Engineering
Concordia University
Montreal, QC, Canada

Abstract

Models are the main artifacts in Model Driven Engineering (MDE). Hence, the quality assessment of models is an important issue in MDE. Using pattern languages, while building software in the MDE approach, is of special interest to designers. Two major issues in using a pattern are "what pattern to choose?" and "how to apply the selected pattern to have a consistent model?" These issues have direct impact on the quality of models and should be given due attention.

In this paper, we discuss how the idea of supporting patterns in MDE can be viewed as part of an overall verification process. Then, we present one of the core processes that can be used for verification of the application of a pattern language in a UML design. Our process is based on a UML profile defined for Fowler's "Patterns of Enterprise Application Architecture." Finally, we show how the process can be integrated into a modeling tool and help the designer in designing more consistent models.

Keywords: Model Quality, Model Driven Engineering, Pattern Language, Verification, UML

1 Introduction

Producing high quality software has always been an issue for software engineering community. Model Driven Engineering (MDE), as a new paradigm in software engineering, faces this issue as well. Since models are the main artifacts which drive software development in MDE [4], the quality of models is important.

Using pattern languages, while building software in MDE approach, is a key tool for designers. One benefit of using patterns is to help designers to communicate their idea. The term *pattern language* refers to the fact that patterns create a vocabulary, i.e., a common language, used by designers [6].

There are two major issues in using a pattern language. First, "what pattern to choose?" and second, "how to apply the selected pattern to have a consistent model?"

[1] This work is partially supported by the Ministry of Science, Research, and Technology of I.R.Iran
[2] Email: b_zamani@encs.concordia.ca
[3] Email: gregb@encs.concordia.ca
[4] Email: s_kayha@encs.concordia.ca

doi:10.1016/j.entcs.2009.02.065

These issues have direct impact on the quality of models. That means, selecting a wrong pattern or incorrect usage of a pattern could result in inconsistent and inefficient design and therefore low quality software. The tool assistance for quality management and consistency checking is necessary since merely manual inspection or review is not enough [5]. Further, MDE promotes usage of tools in the hope of minimizing the effort for maintenance of model, as well as maximizing the benefits of modeling [9].

The quality of UML models can be viewed from three different aspects: syntax, semantics, and aesthetic [19]. In UML documents, e.g., UML 2.0 Infrastructure [13], Well-Formedness Rules (WFRs) are defined for validating the abstract syntax and identifying errors in UML models. As a formal language, UML uses the Object Constraint Language (OCL) [14] for expressing WFRs. However, the semantic and aesthetic checks, if described, are explained by natural language since they are contingent on the underlying domain of the model. Here is where CASE tools come into play and help designers in finding the problems and checking the quality of the models.

In this paper, our goal is to discuss an overall process for verification of the application of a pattern language in a design. As a proof of concept, we will focus on models that use Martin Fowler's *Patterns of Enterprise Application Architecture* [6] (henceforth, *Patterns of EAA*) as a pattern language. In addition to defining a UML profile for some EAA patterns, we extend the process called Sign/Criteria/Repair (SCR) [21] which, as part of the whole process, is used for verifying the application of a pattern in a design.

SCR is a process which helps designers find problems in the application of patterns in their design and follow the wizards for repairing the problems. We investigate how the SCR process can be customized for some EAA patterns. As our case study, we have integrated SCR into one of the state of the art modeling tools, ArgoUML [16]. We have defined critics in ArgoUML for six EAA patterns and show how the tool (based on the wizards) can help the designer in repairing the problems and producing more consistent models.

The rest of the paper is organized as follows. In section 2, we briefly introduce the idea of pattern languages and discuss the *Patterns of EAA* as a pattern language. section 3 introduces the overall process of verifying pattern languages as well as the SCR process for detecting problems in using a single pattern. In section 4 the case study of integrating the SCR process into a modeling tool is described. Related works are discussed in section 5, and in section 6, we conclude the paper.

2 Enterprise Applications Pattern Language

The term "Pattern Language" first coined by architect Christopher Alexander [1]. Subsequently, software experts have defined hundreds of patterns as solutions to recurring problems in software design. For describing the structure of the patterns, each pattern author has his/her own pattern form. A pattern form consists of several items including the name, problem, solution, and examples of pattern usage. By documenting patterns and the relationship among them, in fact the pattern author is defining a language, called *pattern language*, that could be used by designers in developing new software systems [2]. If we consider each pattern as a recipe for a solution, a pattern language is a set of recipes for a whole system. Pattern names play a crucial role in a pattern language, because designers can use those names as a vocabulary that helps them communicate more effectively [6].

Among many available sources of documented patterns, the most famous one is the seminal book on design patterns known as "Gang of Four" (GoF) [7] after its four authors. GoF patterns are classified based on two criteria, purpose and scope. Purpose reflects what a pattern does and scope specifies whether the pattern applies to classes or objects.

Our focus in this paper is on Fowler's book: *Patterns of EAA* [6]. Over forty patterns are defined in the book as solutions to recurring problems, which are applicable to the web-based enterprise applications. The set of patterns introduced in the book are related to each other and, considering the recommendations given by the author, they can be used to describe an application as a whole. Therefore, *Patterns of EAA* can be viewed as a pattern language for the design of web-based enterprise applications.

The patterns of EAA are decomposed into three layers, based on the idea of three-tiered architecture for client-server platforms, i.e., presentation, domain, and data source. The presentation layer is responsible for user interface, the domain layer deals with domain logic and business rules, and the data source layer is related to communicating with database of the system.

The *Patterns of EAA* is a well-known source used by designers of enterprise applications, albeit, applying a pattern needs expertise. The novice designers are vulnerable to making mistakes in using patterns. Generally there are two types of questions that a designer encounter when using a pattern language such as Patterns of EAA. First, "which pattern to choose?" second, "how to apply the selected pattern to have a consistent model?" The latter question can be rephrased as "is the design consistent considering the dependencies between patterns?"

Answering the first question is not easy, since there are different alternatives for patterns that can be used in each layer of the application. Even though, there are some recommendations for the designer to decide what patterns to use when designing an enterprise application, however there is no exact solution for a particular design problem. For instance, based on the *Patterns of EAA*, in organizing the Domain Logic, three alternatives are available. Table 1 summarizes the discussions given in the Fowler's book in terms of advantages and disadvantages of each pattern from the Domain Layer. According to the table, one of the most important factors in choosing the right pattern for structuring the domain, is the complexity of the domain logic. Unfortunately, there is no metric for measuring this complexity. One solution is to ask an expert to review the requirements and give you a judgment.

Finding the answer for the second question is also challenging, since although the patterns seem to be independent, they are not totally isolated from each other. When working with a more accurate example of a pattern language, the dependencies

Table 1
Alternative Patterns for Domain Layer (Adapted from [6]).

Pattern	**Advantages**	**Disadvantages**
Transaction Script	Easy to use and understand for most developers. Easy to build atop a relational DB. A simple procedural models.	Does not fit with the complex business logics. Duplicate code is inevitable.
Table Module	Works well with moderate business logic. Easy for connecting to relational DB.	Does not fit with the complex business logics.
Domain Model	Handles complex business logic in a well-organized way. Matches well with OO paradigm.	Hard to use and understand for non-OO people. Difficult for connecting to relational DB. Object/Relational mappings are needed.

get more and more cohesive. In Figure 1, we show some of the patterns of EAA and their placement in three-layer architecture, as well as the dependencies between the data source layer patterns and the domain layer patterns. For instance, using the Transaction Script pattern for the domain layer, then there are two alternatives for the data source layer, Row Data Gateway pattern and Table Data Gateway pattern. Using Domain Model in the domain layer, there are two conditional alternatives depending on the complexity of the domain model. For simple domain models, using Active Record pattern is recommended, and for complex domain models, Data Mapper pattern is a better choice. There is no specific dependency between the patterns in the presentation layer and the domain layer patterns, however, the tool set which is used for development may affect the pattern which will be used for the presentation layer.

To make our experiments simple and concrete enough, we have selected four patterns from "Data Source Architectural Patterns" including Table Data Gateway, Row Data Gateway, Active Record, and Data Mapper, as well as the Record Set pattern from "Basic Patterns" and Front Controller from "Presentation Layer." Note that in the *Patterns of EAA* book, the Record Set pattern is placed in the category of "Basic Patterns," not Data Source Layer patterns. However, we put it in the Data Source layer for having a neat figure. The selected patterns are highlighted by bold ellipses in Figure 1.

In the following, we summarize the definition of Table Data Gateway and Record Set patterns (given in [6]) which are the target of our discussions in the coming sections.

Table Data Gateway: An object that acts as a Gateway to a database table. One

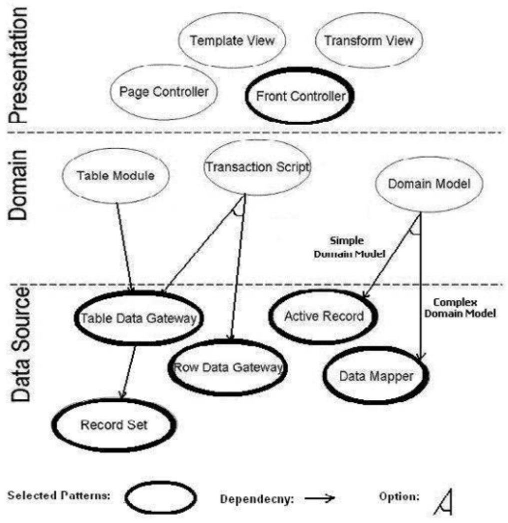


Fig. 1. Some of the Patterns of EAA in a Three-Layer Architecture and the Recommended Dependencies.

instance handles all the rows in the table.

Table 2
The Table Data Gateway Pattern [6].

Person Gateway
find (id) : RecordSet findWithLastName (String) : RecordSet update (id, lastname, firstname, numberOfDependents) insert (lastname, firstname, numberOfDependents) delete (id)

Record Set: An in-memory representation of tabular data.

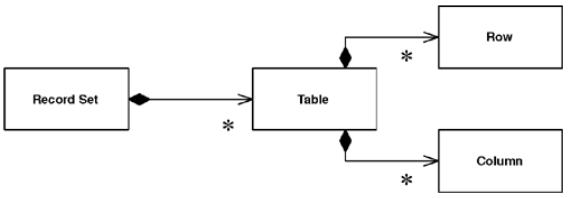


Fig. 2. The Record Set Pattern [6].

As indicated in Table 2, the essence of the Table Data Gateway pattern is that it holds all the SQL commands, e.g., select, insert, update, and delete, in the form of a simple interface, for accessing a single table or view. Others call these methods for interacting with the database. Each method gets the input parameters and maps them into a SQL call which is executed against a database connection. Therefore, the developer does not need to be worried about writing SQL codes.

There are two alternatives to return multiple data items resulted from SQL queries: Map or Record Set. Record Set is another pattern in the EAA pattern language. This is why there is a dependency between Table Data Gateway and Record Set in Figure 1. For people who are familiar with two-tier applications, using a Record Set is more convenient. As Figure 2 shows, the Record Set provides an in-memory structure which is exactly the same as the result of an SQL query. The above discussion shows how the designer is able to utilize a pattern language in designing a system.

3 Pattern Language Verifying Process

Verifying the application of a pattern language in a design is not a single straightforward process. Hence, it is wise to decompose such a comprehensive process into some smaller and simpler subprocesses. One way of decomposition is to first concentrate on verifying the application of single patterns, and then focus on the issues related to the hierarchy of patterns. Furthermore, there are several tasks such as "pattern description" and "pattern language description," that need to be performed inside the process.

The SCR process [21] is one of the core processes for pattern verification which focuses on single patterns. In this paper, we extend the idea of SCR such that as well as verifying the correctness of the application of single patterns, it is able to deal with the issue of dependency between patterns. However, the whole process is still in its infancy. More discussion on the whole process of pattern language verification is given in section 6.

3.1 The SCR Pattern Verifying Process

The SCR process is a simple three-step process for verifying the application of a pattern in a design. The process aims to help the designer, by detecting and fixing the problems in using a pattern. The SCR process consists of the following steps. It is worth noting that this process is based on using UML profiles (See section 3.2).

(i) Sign: The first and most important property of a pattern is its sign. Each pattern has a unique sign. If the process is based on a profile, the sign is simply indicated by a class which has corresponding stereotype. Checking the Sign is the first step of applying the SCR process. If the Sign is present, we continue the process.

(ii) Criteria: The second property of a pattern is a set of criteria that indicates sound and consistent usage of the pattern. If all the criteria are met, a message will be displayed to the designer to inform his/her about using the pattern and stating that the usage of pattern is correct. For each failed criterion, which reflects a problem in the design, a warning message will be reported to the designer.

(iii) Repair: Repair is dependent on the result of criteria evaluation. For correct usage of a pattern, no repair is needed. For problematic or inconsistent usage of a pattern, if there exists a wizard for fixing any of the problems, upon designer's request, the repair takes place and an appropriate message will be displayed to the designer. Otherwise, a message is shown to the designer in order to inform his/her for fixing the problem manually.

Extending the SCR process for verifying the application of a pattern language includes adding more conditions to the Criteria part of the process. These conditions check the dependencies between patterns.

Note that if the process is not based on a profile, slight changes are required in the process. First, there is no explicit sign (as a stereotype) for the pattern. In this case the sign should be considered as part of the criteria that show the structure of pattern. Second, we should consider a threshold for the number of failed conditions in the Criteria step. If the number of failed conditions is less than the threshold then we consider it as a sign for pattern and go to repair step for repairing failed criteria. In this case we call the used pattern a "near-miss." Otherwise, we do not continue the process and suppose that the designer has not used that pattern in his/her design.

Although, the SCR process is not restricted to any specific class of patterns, in the following, we address the problem of applying SCR to the patterns of EAA. But first we need to explain our UML profile.

3.2 A UML Profile for Patterns of EAA

To simplify the detection of patterns of EAA, we exploit profiles, a powerful extension mechanism of UML [13]. A profile can be used to define the mechanisms for tailoring the UML meta model toward specific domains. The notion of profile has matured since its inception. In UML 2.0, a profile could have several parts, including "a set of stereotypes," "a set of tagged values," "WFRs," "a subset of the UML metamodel," and "semantics." Among these, stereotypes are simple but powerful mechanism for extending and adapting UML.

At the present time, our profile consists of a set of stereotypes corresponding to the names of patterns and the names of operations, as indicated in Table 3. In addition to the stereotypes, we have defined WRFs for the profile. In general, the WFRs corresponding to each pattern are the essentials of the pattern. One way of describing WRFs is to use OCL. However, due to the fact that we have defined our profile in ArgoUML, and the tool does not support OCL at the meta model level, we had to hard code the WFRs inside the ArgoUML critics using Java.

As a simple example, the OCL excerpt in the Appendix I-Listing 1 shows the definition of a general-purpose operation which is useful in defining most of the WFRs for patterns of EAA. This OCL defines an operation, named *hasOpSt*, for finding an operation in a class based on its name or its stereotype. This operation, for instance, can be used in another operation that checks whether the context class has four operations for insert, delete, update, and find. As mentioned earlier,

Table 3
The Stereotypes Used in the PofEAA Profile.

Name	**Base Class**	**Description (from [6])**
«tabledata-gateway»	Class	An object that acts as a Gateway to a database table. One instance handles all the rows in the table.
«rowdata-gateway»	Class	An object that acts as a Gateway to a single record in a data source. There is one instance per row.
«activerecord»	Class	An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.
«datamapper»	Class	A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.
«handler»	Class	A controller that handles all requests for a Web site.
«command»	Class	An abstract class which acts as the root for the command hierarchy.
«recordset»	Class	An in-memory representation of tabular data
«insert»	Operation	An operation for inserting data into the database
«delete»	Operation	An operation for deleting data from the database
«update»	Operation	An operation for updating data into the database
«find»	Operation	An operation for finding data in the database
«getter»	Operation	An operation for returning the value of an attribute
«setter»	Operation	An operation for setting the value of an attribute
«doget»	Operation	An operation for receiving get requests from the web server
«dopost»	Operation	An operation for receiving post requests from the web server
«process»	Operation	An operation for carrying out the action related to the command

we have implemented all the WFRs in Java. The Java code corresponding to the above OCL code is shown in the Appendix I-Listing 2. Note that the code uses *Model.getFacade()* as a handler for accessing all the model elements that are related to the current model.

3.3 Adapting SCR for Patterns of EAA

Having defined the stereotypes and the general-purpose operations, let us illustrate how to adapt the SCR process to detect problems in the application of one of the EAA patterns, the Table Data Gateway pattern, and also to check the dependency between this pattern and the Record Set pattern.

(i) Sign: The context should be a class with stereotype «tabledatagateway».

(ii) Criteria: The requirements of a sound Table Data Gateway pattern are as follows.

 a) The class needs operations for insert(), delete(), and update(), and usually consists of several find() operations. Each operation is recognized by its name or stereotype. As it is mentioned in previous section, we are using the UML extension mechanism of stereotypes as an alternate way of detecting an operation. For instance, if the name of operation starts with "insert" or if it has stereotype «insert», then we recognize it as insert() operation. The general-purpose operation *hasOpSt* which is shown in the Appendix I is helpful at this step.

 b) Each find operation should have Record Set as the return type. This condition is in fact for checking the dependency between two patterns which is shown in Figure 1.

 c) The parameter list of insert operation should be subset of the parameter list of update operation.

 If all the conditions are satisfied, the correct usage of the pattern is reported to the designer.

(iii) Repair: According to the above Criteria, there are three possible problems in using the Table Data Gateway pattern (remember that we had three conditions). If any of four operations is missing or if any of the find operations has a return type other than Record Set, then appropriate error message will be displayed to the user. In case there are wizards for fixing the problems, and the user decides to apply the changes based on the wizards, then the appropriate operations will be added to the class or the return types are fixed. In addition to changing the return type of the find operations, an instance of the Record Set pattern will be automatically created in the model. After fixing each problem, an informative message is displayed to the designer to aware him/her of the change. If there is a mismatch between the parameters, i.e., the third criterion is failed, then a warning message will be displayed to the designer to inform him/her of the problem in the design that needs to be corrected manually.

 It should be noted that after all the problems are fixed, the correct usage of pattern is detected and will be reported to the user.

For other patterns, there are small variations in applying the SCR process, briefly described in the following.

- For Row Data Gateway pattern, there is a gateway class which has attributes that match with columns in database table, and there is a finder class which uses this gateway to access every record of the database. The result of the find operation in Row Data Gateway is a record instead of a table.

- Active Record pattern wraps a row in a table, therefore it has one attribute for each column of the database table in addition to all above mentioned operations. The Criteria should contain conditions that check the structure of the pattern.
- Data Mapper pattern is used to move data between objects and a database, therefore it has the above mentioned operations but no attributes.
- Record Set pattern contains classes for table, row, and column with containments as it is indicated in Figure 2. The Criteria check the structure of the pattern.
- Front Controller pattern consists of a handler class with a dependency to an abstract class which has the structure of the Command pattern.

4 Integrating SCR into Modeling Tools

To integrate the SCR process into an IDE, we need to have facilities for describing the Sign, the Criteria, and the Repair, and be able to invoke each of these parts from the IDE. Instead of building such environment, we decided to integrate the SCR process into existing tools. As our case study, we have selected ArgoUML, an open source state of the art modeling tool. In this section, we provide a brief overview of the tool, we discuss the implementation aspects of the SCR process, and we give an instant evaluation of the suitability of the tool for the SCR process.

4.1 ArgoUML

ArgoUML [16] is an open source UML modeling tool that supports all standard UML 1.4 diagrams. Besides features such as diagram editor and reverse engineering of compiled Java code, ArgoUML is a design critiquing tool. As the creator of ArgoUML defines "A *design critic* is an intelligent user interface mechanism embedded in a design tool that analyzes a design in the context of decision-making and provides feedback to help the designer improve the design" [15].

Simply put, ArgoUML has predefined agents, called critics, that are constantly investigating the current model and if the conditions for triggering a critic hold, the critic will generate a ToDo item (this item is called a *critique*) in the ToDo list. A ToDo item contains a short description of the problem, some guidelines about how to solve the problem, and if there exists, a wizard which helps the designer solve the problem automatically.

The critics run as asynchronous processes in parallel with the main ArgoUML tool. The critics are not intrusive, since the user can totally ignore them or disable one or all of them by the critics' configuration menu. The critics are not user defined, since they all are written in Java and are compiled as part of the tool. Furthermore, a ToDo item generated by a critic will remain in the ToDo list until the origin of the problem is vanished, either manually by the designer or by following the wizards proposed by the tool.

4.2 Integrating SCR into ArgoUML

In order to integrate the SCR process into ArgoUML, we have benefited from both Robbins' Ph.D. thesis [15] and the ArgoUML Cookbook [17]. Our implementations are divided into the following two parts.

First part deals with detecting wrong pattern usages. For each pattern, we write a critic class that if the Sign of the pattern is found, then it checks the Criteria. If any of the criteria is failed, the critic is triggered and a ToDo item (a critique) will be posted in the ToDo List. We have created a new section called **Patterns of EAA** under category "By Decision" in the ToDo List of the ToDo pane of ArgoUML. Furthermore, we have written a wizard class for doing Repair part of the SCR process for each pattern. By selecting a ToDo item, the description of ToDo item will be shown in the Details pane, and upon the user's request, the wizard for the critic will be executed and the problems found in the pattern usage will be fixed.

Second part of the implementations is for finding correct usage of a pattern and reporting it to the designer. For this purpose, a new tab called **Detected Patterns** is added to the Details pane of ArgoUML. Correct usages of patterns will be shown in a tree-like format in this tab with two main branches: **Design Patterns** and **Patterns of EAA**. The Design Pattern branch is dedicated to the GoF patterns which are not the target of our discussion in this paper. By detecting a pattern in each branch, the name of the pattern is shown in the tree and by selecting the pattern, all classes that play a role in the pattern will be displayed.

Figure 3 in the Appendix I shows the tool interacting with the designer in fixing the problems found in the application of the Table Data Gateway pattern. Part a of the figure shows how a ToDo item is inserted in the ToDo List after the critic is triggered (because of missing operations). Part b shows the options that are given to the user as a wizard to add missing items. Part c shows how the problems are solved automatically and the missing model elements are added to the class. Part d shows that after fixing the problems in the pattern, the correct pattern is automatically detected and is shown in the Detected Patterns tab.

Some major points for evaluating the extensibility of ArgoUML are as follows. From the one hand (positive side), we believe that the concept of "design critiquing," as the main idea behind generating ArgoUML, is extremely compatible with the idea of "verifying the application of a pattern language in a design" as the main idea of our research. Hence, ArgoUML is an appropriate platform for integrating the SCR process. In addition, the ToDo items and the Wizards in ArgoUML are very interactive and user friendly and there is much hope that we can encapsulate the knowledge that is embedded between the lines of the Patterns of EAA book into these parts of the tool.

From the other hand (negative side), firstly, due to the fact that ArgoUML critics are implemented in Java and adding new critics requires Java expertise and is done by modifying the source code, end-users cannot add new critics for criticizing new patterns. Secondly, it is not possible to define critics using OCL in ArgoUML, since in ArgoUML the OCL constraints can be written at the model level only. And last but not least, an important disability of ArgoUML in applying the SCR process is dependent on the logic behind critics. The fact is that critics are triggered only when one of the Criteria is violated. That means, critics are always trying to criticize a design, not to confirm its correctness. Hence, there is no possibility to inform the user about *correct usage of a pattern* in ArgoUML. As mentioned earlier, we have solved the last problem by looking for the correct usage of the pattern along with the criticizing process and by reporting the correct patterns in a new tab.

5 Related Work

There exist several works related to the verification of a design. Some works focus specifically on detecting GOF design patterns, while others intended to work on quality assessment of models.

In the area of detecting GOF patterns, Bergenti and Poggi [3] have built a tool which uses Prolog rules to detect detectable GOF design patterns (only eleven patterns are considered detectable). The tool is integrated into ArgoUML and is claimed to be extensible and customizable. Heuzeroth et al. [8] have built a Java analyzer tool which detects some of the GOF design patterns in legacy code. They have coded the pattern essentials into their analyzer. Roel [20] has built a declarative framework for reasoning about Smaltalk code. He has defined Prolog rules that checks structure of a pattern inside a program. Tsantalis et al. [18] have built an automatic design pattern detector based on a similarity scoring algorithm. They use a matrix format to capture the essence of each GOF pattern, and by converting the class diagram of the given system into a set of matrices, find pattern matches in the design.

In the area of model quality assessment, Breu and Chimiak-Opoka [5] have introduced a framework for quality assurance of models based on the concepts of Queries, Checks, and Views. Liu et al. [11] have introduced a classification for design inconsistencies, and then they have developed an expert system (a rule-based system) using JESS for detecting inconsistencies in a given UML design. The system is able to give advice to the user and fix the problems automatically.

Kolovos et al. [10] have defined a language named Epsilon Wizard Language (EWL) which has concepts very close to the SCR process. Each wizard has a title, a guard, and a body. The brilliant aspect of their work is that the user can define the specifications of wizards off line in EWL language. As a tool, EWL is integrated into ArgoUML. That means, by running ArgoUML and selecting model elements, if the guard of a wizard is true, the title is displayed to the user and the body part is executed. EWL has model modification capabilities that can be used in the body part of wizards to repair the problems in a model.

Robbins [15], in his Ph.D. thesis has described several cognitive theories relevant to software engineering. Furthermore, he is the developer of ArgoUML, which seems to be the most comprehensive design critiquing system presented until now.

In [21], we have compared the suitability of three modeling tools, ArgoUML, EWL, and OCLE [12], regarding the integration of the SCR process.

6 Conclusion

The main aspects of verifying the application of a pattern language for a domain, e.g., the *Patterns of EAA*, are as follows.

- To detect whether a pattern is used correctly or wrongly?
- To detect the "near-misses" of a pattern?
- To detect whether the design is consistent, in terms of used patterns? I.e., to detect problems in the structuring mechanism of the pattern language.
- To help the designer in repairing the problems that are found in the application of patterns.

While full support for this verifying process needs a lot of work, in this paper, we have addressed all above aspects in part. We introduced a UML profile for easing the detection of patterns. We refined the already presented process named Sign/Criteria/Repair (SCR) as a core process which aims in detecting and repairing the problems in the usage of a single pattern as well as checking the dependencies between patterns. A Sign is the basic characteristic of a pattern, usually in the form of stereotypes. Criteria are the minimal requirements of the correct usage of a pattern. Repair is a set of steps to fix problems in the application of pattern. Each pattern has specific Sign and Criteria. Each problem has essential Repair steps. The refinement to SCR includes adding more conditions to the Criteria section for checking the consistency of patterns.

To evaluate the idea of SCR and its applicability and usefulness in current modeling tools, we did experiments with the ArgoUML modeling tool. We observed that the SCR process is able to be integrated in ArgoUML by writing Java code, and it helps designer in detecting problems early in the design process. However, hard coding the process steps into the tool is not a convenient way of tool extension.

As part of our future work, we aim to work on the whole process of verifying the pattern language application in a design. Completing the pattern profile, and guiding user in selecting the right pattern are parts of the whole process. As the completion of our work with ArgoUML, we plan to implement critics and wizards for all patterns of EAA and encapsulate the knowledge that is embedded between the lines of the pattern books into Wizard and ToDo parts of the tool to guide the user for selecting the right pattern. This support for verification of models, would help designers to see problems is their design, how far they are from a sound design, and how much progress they have made in fixing the problems. For the repair part, having the possibility of preview, such as what is available for code refactoring in IDEs would be another nice feature.

References

[1] Alexander, C. et al., "A Pattern Language: Towns, Buildings, Construction," Oxford University Press, 1977.

[2] Berczuk, S., *Finding solutions through pattern languages*, Computer **27** (1994), pp. 75–76.

[3] Bergenti, F. and A. Poggi, *Improving uml designs using automatic design pattern detection*, in: *12th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2000, pp. 336–343.

[4] Bézivin, J., *Model driven engineering: An emerging technical space*, in: *Proceedings of Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005*, Lecture Notes in Computer Science **4143** (2006), pp. 36–64.

[5] Breu, R. and J. Chimiak-Opoka, *Towards systematic model assessment*, in: *Perspectives in Conceptual Modeling*, Lecture Notes in Computer Science **3770** (2005), pp. 398–409.

[6] Fowler, M., "Patterns of Enterprise Application Architecture," Addison-Wesley Professional, 2002, 1st edition.

[7] Gamma, E., R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[8] Heuzeroth, D., T. Holl and G. Hogstrom, *Automatic design pattern detection*, in: *11th IEEE International Workshop on Program Comprehension (IWPC'03)* (2003), p. 94.

[9] Kent, S., *Model driven engineering*, in: *Proceedings of Third International Conference on Integrated Formal Methods (IFM)*, Lecture Notes in Computer Science **2335** (2002), pp. 286–298.

[10] Kolovos, D. S., R. F. Paige, F. A. Polack and L. M. Rose., *Update Transformations in the Small with the Epsilon Wizard Language*, Journal of Object Technology (JOT), Special Issue for TOOLS Europe **6** (2007), pp. 53–69.

[11] Liu, W., S. Easterbrook and J. Mylopoulos, *Rule-based detection of inconsistency in uml models*, in: *Workshop on Consistency Problems in UML-Based Software Development*, Dresden, Germany, 2002, pp. 106–123.

[12] OCLE, *Object contraint language environment official web site*, http://lci.cs.ubbcluj.ro/ocle/ (Retrieved on May 20, 2008).

[13] OMG, "Unified Modeling Language: Infrastructure, v2.0," OMG document formal/05-07-05, 2005.

[14] OMG, "Object Constraint Language: Specification, v2.0," OMG document formal/06-05-01, 2006.

[15] Robbins, J. E., "Cognitive Support Features for Software Development Tools," Ph.D. thesis, University of California, Irvine (1999).

[16] Tigris.org, *Argouml official web site*, http://argouml.tigris.org/ (Retrieved on May 20, 2008).

[17] Tolke, L., M. Klink and M. van der Wulp, "Cookbook for Developers of ArgoUML," University of California, http://argouml-stats.tigris.org/documentation/defaulthtml/cookbook/, 2007-01-16 edition (Retrieved on May 20, 2008).

[18] Tsantalis, N., A. Chatzigeorgiou, G. Stephanides and S. T. Halkidis, *Design pattern detection using similarity scoring*, IEEE Trans. Software Eng. **32** (2006), pp. 896–909.

[19] Unhelkar, B., "Verification and Validation for Quality of UML 2.0 Models," Wiley-Interscience, 2005.

[20] Wuyts, R., *Declarative reasoning about the structure object-oriented systems*, in: *Proceedings of the TOOLS USA '98 Conference* (1998), pp. 112–124.

[21] Zamani, B. and G. Butler, *Critiquing the application of pattern languages on uml models*, in: *Workshop on Quality in Modeling, MODELS2007 Conference*, Nashville, TN, USA, 2007, pp. 18–35.

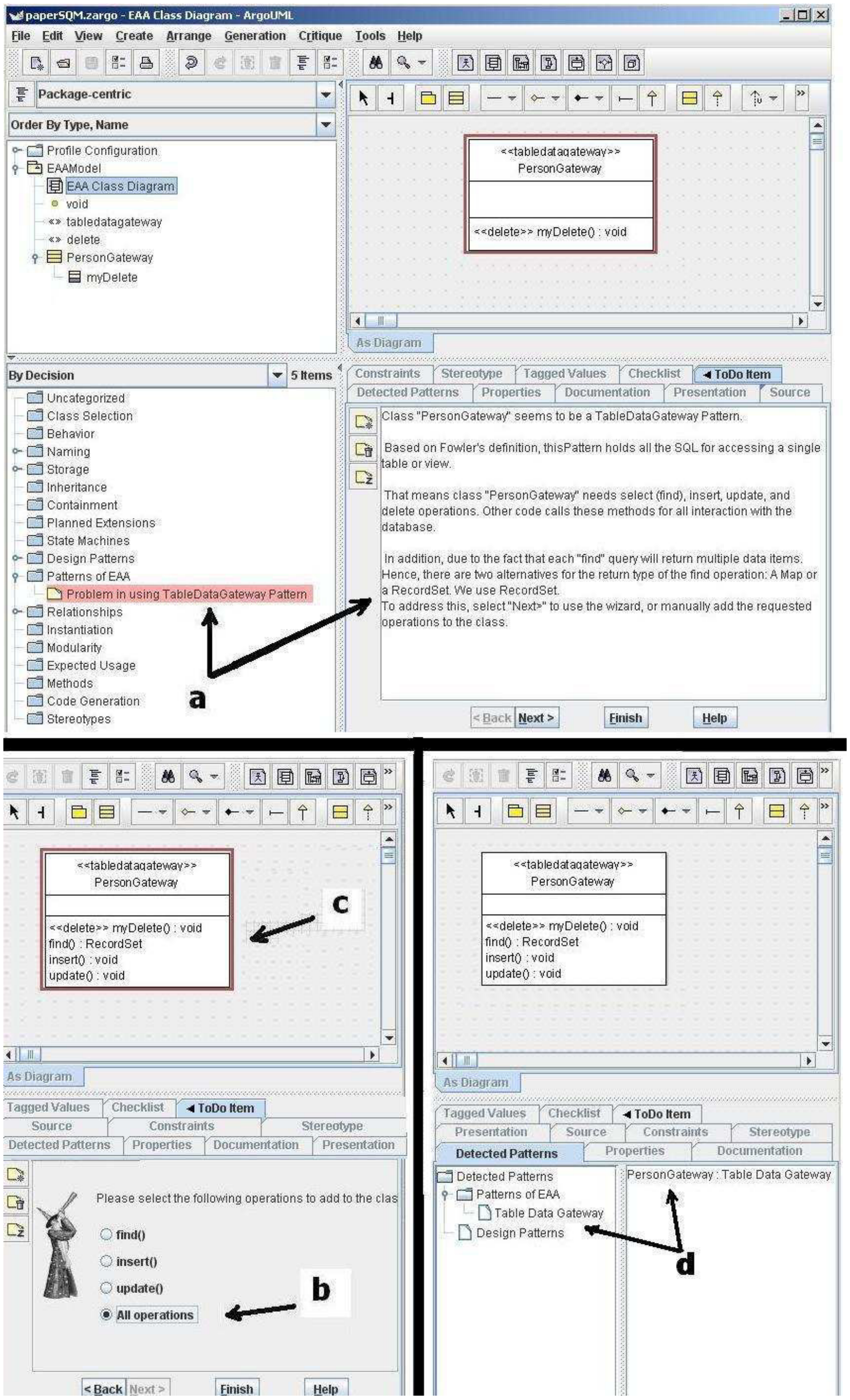
Appendix I

Listing 1. Defining an operation in OCL

```
context Class
def: let hasOpSt(op:String) : Boolean =
     Operation.allInstances ->
      exists( o:Operation | o.owner = self and
      ((o.name.substring(0,op.size()-1) = op) or o.hasSt(op)))
```

Listing 2. Defining an operation in ArgoUML (using Java)

```
public static boolean
  hasOpSt(Object cls, String op) {
  boolean found = false;
  Iterator operator = Model.getFacade().getOperations(cls).iterator();
  while (operator.hasNext()) {
    Object o = operator.next();
    String opName = Model.getFacade().getName(o);
    if (opName.startsWith(op))
      { found = true; break; }
    Iterator s = Model.getFacade().getStereotypes(o).iterator();
    while (s.hasNext()) {
      String sName = Model.getFacade().getName(s.next());
      if (sName.equals(op))
        { found = true;  break; }
    }
    if (found) break;
  }
  return found;
}
```



Class "PersonGateway" seems to be a TableDataGateway Pattern.

Based on Fowler's definition, thisPattern holds all the SQL for accessing a single table or view.

That means class "PersonGateway" needs select (find), insert, update, and delete operations. Other code calls these methods for all interaction with the database.

In addition, due to the fact that each "find" query will return multiple data items. Hence, there are two alternatives for the return type of the find operation: A Map or a RecordSet. We use RecordSet.
To address this, select "Next>" to use the wizard, or manually add the requested operations to the class.

Please select the following operations to add to the clas

find() [ ]
insert() [ ]
update() [ ]
[✓ All operations]

Fig. 3. Screenshots of applying the critic for the Table Data Gateway Patten in ArgoUML.
(a) Critic is triggered due to problems; A ToDo Item is created in ToDo list.
(b) Wizard is activated since the user decided to fix the problems using wizard.
(c) User has allowed the wizard to fix all the problems; The missing elements are added to the model.
(d) Correct usage of patterns are reported in the “Detected Patterns” tab in the Details Pane.