Tool Support for Verifying Applications using Object-Oriented Patterns

Gabriela Aranda, Andrés Flores, Agustina Buccella and Luis Reynoso
Departamento de Informática y Estadística, Universidad Nacional del Comahue,
Buenos Aires 1400, Neuquén, Argentina
phone: (54) 299 - 4490312, fax: (54) 299 4490313
Email: garanda,aflores,agusb,lreynoso@uncoma.edu.ar

Abstract. Applying design patterns is considered a helpful technique for designing software systems. Patterns description, however, results not sufficiently precise providing a weak understanding and making it difficult to be certain when a pattern is being used correctly. We have formally specified properties of patterns and a formal basis for object-oriented design. In the present work, our formal basis is used as a support to an automatic tool for verifying proper applications of patterns. The usage of this tool is mainly focused on the learning process about patterns applications. Through a better understanding of patterns, the designer may certify when and how a pattern is being appropriately applied to solve a specific design problem. Furthermore, the whole design process could be improved by using a precise technique supported by an automatic tool for verification.

1. Introduction

Designing software systems can be assisted by different design techniques such as applying design patterns, which is one of the most useful techniques because their widely recognised advantages in reinforcing several quality attributes such as high modifiability and reusability. Patterns are abstractions of concrete design problems which recur in a range of different contexts [1], and they describe a generic solution to these problems, which can be used to obtain a solution to a specific problem [2,3]. GoF catalogue [4] presents a well-known group of design patterns which uses object-oriented notation to capture the experience of several experts in software design.

Patterns are described by means of natural language narrative and graphical notation, which gives a sort of abstraction allowing a wide range of usage. However, this kind of notation results not sufficiently precise providing ambiguities and inconsistencies which leads a weak understanding and makes it difficult to be certain when a pattern is being used correctly. Following the goal of Pattern-Based Design, there should be applied the design principle of *rigour and formality*, in order to state a solution and to enhance such quality attributes as well. With this in mind, we have developed in the RAISE Specification Language RSL [5], a formal basis for object-oriented design where patterns may be applied [6] and formally specified the properties of each pattern in the GoF catalogue [7,8,9]. Thus, providing a more precise notation can improve understanding about patterns and designers can grow in the knowledge of when and how a pattern is being applied appropriately.

An enhancement concerning flexibility might be added to the process of pattern-based design. For this, we intent to build an automatic tool for modelling object-oriented design with patterns. The formal basis gives the precise format in which a design model can be represented with the aim of providing facilities for verifying a proper application of a pattern. The designer can graphically model a design specifying a particular pattern that suits with the problem. Thereafter, verification activities are performed on the representation of the graphical model.

Since our formal model was specified at an abstract level, it should be translated into a more concrete one in order to be closer to the codification phase. Thus, we have decided to translate the RSL Specification to an object-oriented model maintaining the structures and semantic of the building blocks. The representation of the graphical model in the tool is translated to an object structure according to the building blocks of the formal model in order to run the verification tasks.

The usage of this tool is also focused on the learning process about pattern application. Through a better understanding of patterns, the designer may certify when and how a pattern is being appropriately applied to solve a specific design problem. Furthermore, the design process may be improved with a precise technique supported by an automatic tool for verification.

We briefly present the formal basis in section 2 and its translation into an object-oriented model in section 3. Then, in section 4, the structure for the automatic tool is presented. We discuss future work and conclusion afterwards.

2. RAISE Model of a Pattern-based Design

Pattern-based design involves the binding of pattern elements to elements of the design [10]. A subset of the classes and relations in a design then conforms to a specific pattern if their properties are the same as those of the counterparts in the pattern. Our formal model was specified according to a Bottom-Up approach such that, given a (subset of a) design, find a pattern that matches it or check if a given pattern matches it [7]. For its definition the properties of a general object-oriented design were abstracted out. It includes the components of OMT-extended notation: classes, methods, variables, and relations. In addition, we have specified the meaning behind the notation: hierarchical properties, meaningful relationships between variables and relation names, and description of method's functionality.

Building blocks of the formal model where specified by using the RAISE specification language RSL[5]. Following we briefly explain its main constituents. See [6] for details and [11] for full specification.

- **Design Structure**: A design consists of a collection of classes and a collection of relations.
  Design_Structure = C.Classes x R.Wf_Relations
- **Classes**: Each class has a name, which is unique in the design, a set of methods, a state which is represented as a set of variables, and a type which may be concrete or abstract.

```
Design_Class :: class_state : G.State
                class_methods : M.Class_Method
                class_type : G.Class_Type
```

- **Methods**: Every method has a name and a list of parameters. Also a pseudocode annotation attached to the method indicating the actions it will perform. Generic actions can be included in an annotation. They are mainly: Invocation, Instantiation and Conditional structure. See [11] for a detailed specification.
- **Relations:** A relation is basically determined by the classes it links and its type, which may be inheritance, association, aggregation, or instantiation. All relations are represented in the model as binary relations. We have modelled source and sink classes as well as their cardinality.

```
Design_Relation :: relation_type : Relation_Type
                   source_class : G.Class_Name
                   sink_class : G.Class_Name,

Ref ::
   relation_name : G.Wf_Vble_Name
   sink_card : G.Card
   source_card : G.Card,

Relation_Type ==
   inheritance | association(as_ref: Ref) |
   aggregation(ag_ref : Ref) | instantiation,
```

- **Design-Pattern Binding**: In order to link a subset of a design model to a specific pattern there is a *renaming map*, which associates design entities (classes, state variables and methods) with corresponding entities in the pattern. This mapping indicates the *role* each entity in the design plays in the pattern. Then a design matches a particular pattern if all the entities in the design

playing a pattern role satisfy its properties. Pattern properties where specified as Boolean-valued functions (see [9,11 ,12,13 ,14] for examples).

```
ClassRenaming ::
    classname : G.Class_Name
    methodRenaming :
        Method_and_Parameter_Renaming
    varRenaming : VariableRenaming,

Renaming =
    G.Class_Name -m-> ClassRenaming-set,
```

The formal model as a whole then includes the *Design Structure* and the *Renaming* mapping, in such a way that given a design and a renaming to a specific pattern it is possible to verify if there is a matching.

```
Design_Renaming = DS.Wf_Design_Structure x Wf_Renaming,
```

3. Moving from RAISE Specification to an Object-Oriented Model

Every building block in our formal model will have a counterpart in the object-oriented model. Figure 1 presents an example: the *Wf_Design_Structure* type will be represented by a class called by a similar name. Since the *Design_Structure* type comprises a collection of *Classes* and a collection of Relations (*Wf_Relations*), there will be two *state variables* in that class, which will be used to relate this class to others, with appropriate names, representing the collection of Classes and the collection of Relations.

```
Design_Structure = C.Classes x R.Wf_Relations,
Wf_Design_Structure = {| ds : Design_Structure :- is_wf_design_structure(ds) |}
```

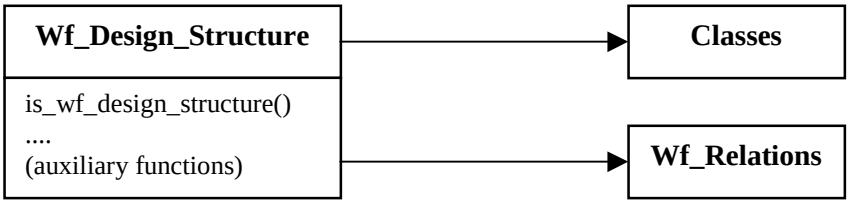

Figure 1: *Translation of a RAISE type to OOModel*

Since the principal goal of this work is to verify correctness in a design model there is a function in every building block of the formal model whose purpose is checking well-formedness. For example for the *Design_Structure* to be checked if it is well-formed there is a function called *is_wf_design_structure*. This function is represented as a method in the class *Wf_Design_Structure* of the object-oriented model. Functions called by the previous one are represented as private methods in the same class. For every building block there are "*auxiliary functions*" allowing verification but mainly supplying a shortcut through the whole structure. Those functions are methods included in the interface of the class representing that building block. The same process is applied for the specification of every GoF pattern.

4. Verification Tool for OOD using GoF Patterns

Basically the tool is divided into two layers: the modelling layer, whose result is a specification of an object-oriented design model provided by a graphical component; and the verification layer, which carries out the process of checking the correctness of the design model and also if the design subset related to a specific pattern satisfies the pattern properties.

4.1. Specification Layer

Since many tools providing a graphical component for modelling object-oriented design have already been developed with a proved success, we have decided to choose one of them, and thus concentrate the major effort in the field where less work has been delivered. After studying some tools available in the market we have selected a non-commercial tool called *FUJABA* [15]. This tool was developed in Java and it may produce a Java specification of the object-oriented design that was modelled by using it.

The functionality of *FUJABA* is quite similar to our expectations. However, according to our formal basis, a new behaviour needs to be developed in order to be able to represent an entire object-oriented model in which a pattern has been applied.

The extended functionality is related, for example, to the *annotations* attached to methods of classes. Annotations are used to express how collaborations between classes are carried out. Since our formal model expresses collaborations in a static way, annotations come to fulfil this subject. Other important new behaviour concerns the possibility of selecting a particular pattern from a pattern repository and setting which pattern roles are played by different entities at design level.

Some changes are also necessary, mainly in the notation of the object-oriented model , given by a Java specification. An example is the simplification needed when an *aggregation relation* is represented, thus complexity of the grammar for a Parser is not increased. Pattern roles and other design or pattern elements will be expressed adding "*comments*" in the Java specification.

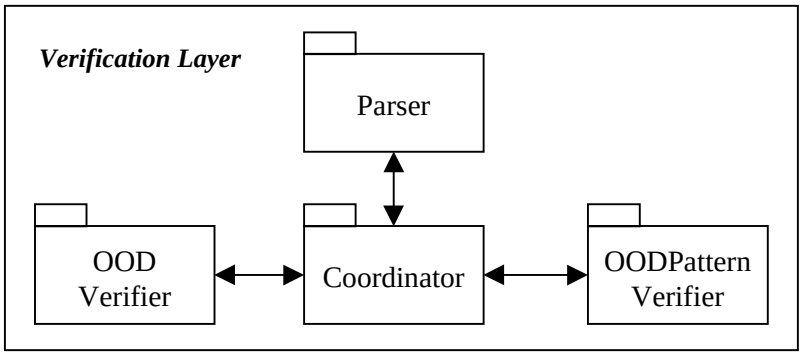

Figure 2: *Verification Layer Components*

4.2. Verification Layer

This Layer is composed of some components co-ordinated by a *Coordinator* module to deal with verification activities. The rest of components are: a *Parser*, a *OOD Verifier*, and a *OODPattern Verifier* as is showed in Figure 2.

Basically the Java specification obtained from the graphical component will be the input for the *Parser* component, which will create the representation of the model according to our formal model of an object-oriented design. As a result, an object structure, will be passed to the *OOD Verifier* component. This component will verify, by using the appropriate methods of the object structure, the correctness at design level of the whole model. In case an error is detected, then this component will return where the error was found in such a way it can be fixed in the graphical component.

If the result of the *OOD Verifier* is successful, then the *Coordinator* will pass the object structure to the *OODPattern Verifier*. This component will check if the properties of the selected pattern (in the graphical tool) are in fact satisfied. Once again, if an error is found the result will tell the designer what the property not satisfied is.

5. Conclusions

The main goal of our work is to accomplish a more precise but flexible Pattern-Based Design process. For this, we have developed, by means of RAISE [5], a formal basis of an object-oriented design and GoF design patterns [4], which was presented in section 2. In order to provide flexibility we are developing an automatic tool for verifying a correct application of patterns. With this in mind, the abstract formal model, giving the structure for representing a verifiable design model, needs to be concretised. Thus, a translation from the RAISE specification to an Object-Oriented Model is being carried out (see section 3). The main aspects concerning the automatic tool including its structure and its main components have been introduced in Section 4. Very often more than one pattern is needed to solve a design problem. If indeed our tool verifies a single pattern at a time, a piece of work has been already done concerning composition among patterns in [14]. Our current work is intended to be applied, in future, to improve a component development process.

6. References

1. Brad Appleton. *Patterns and Software: Essential Concepts and Terminology*. http://www.enteract.com/~bradapp, November 1997.
2. Doug Lea. *Patterns-Discussion FAQ*. http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html, December 1999.
3. Robert Zubeck. *Much Ado about Patterns*. http://www.acm.org/crossroads/xrds5-1/patterns.html, March 2000.
4. Gamma E., Helm R., Johnson R. and Vlissides J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
5. The RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series, Prentice Hall, 1992.
6. Andres Flores, Luis Reynoso and Richard Moore. *A Formal Model of Object Oriented Design and GoF Design Patterns*. In proceedings of the FME 2001, Formal Methods Europe, Berlin, Germany, LNCS 2021, Springer Verlag 2001, 12-16 March 2001, pp. 223-241.
7. Andres Flores and Richard Moore. *Analisis and Specification of GoF Structural Patterns*. In proceedings of 19th IASTED, International Conference on Applied Informatics (AI 2001), Innsbruck, Austria, 19-22 February 2001, pp. 625-630.
8. Luis Reynoso and Richard Moore. *A Precise Specification of GoF Behavioural Patterns*. In proceedings of SNPD'01, 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing, Nagoya, Japan, 20-22 August 2001, pp. 262-270.
9. Gabriela Aranda and Richard Moore. *GoF Creational Patterns: A Formal Specification*. Technical Report 224, UNU/IIST, P.O. Box 3058, Macau, 2000.
10. Marco Meijers, *Tool Support for Object-Oriented Design Patterns*. Master Thesis. Department of Computer Science, Utrecht University, The Netherlands, http://www.serc.nl/people/florijn/work/patterns.html, August 1996.
11. Andres Flores, Luis Reynoso and Richard Moore. *A Formal Model of Object Oriented Design and GoF Design Patterns*. Technical Report 200, UNU/IIST, P.O. Box 3058, Macau, July 2000.
12. Andres Flores and Richard Moore. *GoF Structural Patterns: A Formal Specification*. Technical Report 207. UNU/IIST, P.O. Box 3058, Macau, August 2000.
13. Luis Reynoso and Richard Moore. *GoF Behavioural Patterns: A Formal Specification*. Technical Report 201, UNU/IIST, P.O. Box 3058, Macau, May 2000.
14. Gabriela Aranda and Richard Moore. *Formally Modelling Compound Design Patterns*. Technical Report 225, UNU/IIST, P.O. Box 3058, Macau, December 2000.
15. Software Engineering Group. *FUJABA (From UML to Java And Back Again).* University of Paderborn, Germany, http://www.uni-paderborn.de/cs/fujaba.