Structural Modeling of Design Patterns: REP Diagrams

José Luis Isla Montes[1], Francisco Luis Gutiérrez Vela[2]

[1]Departamento de Lenguajes y Sistemas Informáticos – University of Cádiz
E.S. Ingeniería – c/ Chile, 1 – C.P. 11002 – Cádiz – España

[2]Departamento de Lenguajes y Sistemas Informáticos – University of Granada
E.T.S.I. Informática – Periodista Daniel Saucedo Aranda s/n – C.P 18071 – Granada – España

joseluis.isla@uca.es, fgutierr@ugr.es

Abstract

Software modelling languages should possess a notation, semantics and treatment adapted to design patterns, but its correct specification is a challenge for many investigators. UML treats them as parameterised collaborations, however, this approach is not exempt of problems. The goal of this work is the elaboration of a simple and intuitive model for the structural specification of design patterns and its integration in UML. The patterns specified with this model expect to be true reusable templates that can be applied in different contexts. For that, we use a visual, complete, simple and easy to learn notation. We believe that this model could be used successfully for the construction of a future tool that facilitates the definition, application, visualization and validation of patterns.

1 Introduction

Design patterns have acquired great popularity among researchers and software designers. The reason of their great success is that they form a common vocabulary of good solutions applicable to typical design problems that can appear in different contexts. Design patterns facilitate the reuse of the expert knowledge as “design components", improving the documentation, understanding and communication of the final design.

We think that the software modeling languages should possess an appropriate notation, semantics and treatment that facilitates the use of the design patterns as modeling elements of first order. On the other hand, the tools of software design based on these languages should facilitate their definition, application and validation in specific contexts.

In the following section we will summarize several important works that have treated the specification of design patterns and we will justify the interest of this new approach. In the third section we will show some of the inconveniences that UML presents to represent the essence of a pattern appropriately. Next, in the fourth section, we will try to summarize the new philosophy of the specification model, defining the main concepts and its relationships. As an example, in the section fifth, we show the application of the new approach for the specification of the Abstract Factory pattern [5]. In the last section we will present the conclusions, as well as the questions that are still to explore.

2 Works about design patterns specification

The description that Gamma et al. [5] make of their patterns is good, however, it is mainly narrative and it is based on concrete examples expressed with OMT. These are obstacles for the treatment with a tool.

It is necessary to find a notation that has the capacity to specify, in a precise way, the essence/invariant of a pattern. That is to say, the group of structural and behavioural restrictions that characterize it and whose instances it will have to satisfy.

Several interesting works exist in this line, among these it is necessary to highlight:

- The formal language "LePUS" (**L**anguag**e** of **P**atterns **U**niform **S**pecification) [1][2][3][4] which abstracts from the implementation elements (objects, attributes, methods, etc.). It uses logical expressions and a semantically equivalent visual notation starting from a simple universe where the classes and the functions are atomic entities. LePUS allows to define relationships among groups of these entities (inheritance, reference, etc.), which should exist in a program or model to be able to conform with the specification.
- According to Lauder and Kent [8], the specification of a pattern should be divided in three levels (roles, types and classes) ordered from the greater to lesser level of abstraction. The first level captures the essence of the pattern obviating specific details of the application domain, the second refines the previous one usually adding specific restrictions of the domain and the last represents a concrete implementation. A visual notation denominated “constraint diagrams” [7] is used for the precise modeling of the static and dynamic structure of the patterns. These are used in combination with another modeling graphic notation, in this case UML.
- The “Fragment Model” of Meijers [10] in which a pattern is divided in fragments connected by roles. A fragment would be each one of the outstanding components of a pattern and a role, a property of a fragment, would connect with the fragments that are actors of that role.
- The “Role Diagrams” of Riehle [12] where the patterns are represented as collaborations among roles (a specific view of the responsibilities of an object) as abstraction from the class diagrams.
- Lastly, the interesting works of Le Guennec et al. [9] and Sunyé et al. [13] centered on UML. They modify the metamodel to use stereotyped occurrences of patterns with OCL restrictions in the meta-level and they use the base of the stereotype as context (the metaclass PatternOccurrence). Their ideas are transferred to the UMLAUT tool.

In this work we present an alternative model of specification for the patterns, based on what we denominate REP Diagrams. Contrary to the other approaches, with this model it is possible to carry out a precise and visual specification for the structure of a pattern by means of a simple extension to UML, without necessity of appealing to a formal notation such as OCL. The result is a familiar, simple, complete and very easy to learn notation for the designer and a model that treats the design patterns as true templates. For these reasons, we believe that the construction of a future tool based on this model would be of a great interest.

3 The case of UML

As it is already known, UML [11] treats design patterns as parameterised collaborations, however, this treatment presents some limitations to express the semantics associated to a pattern correctly, mainly because initially these have different purposes.

A collaboration simply seeks to model the interaction among a group of instances to achieve a specific task. However, a pattern has a more general purpose, it should specify what the structure and behaviour of a design solution is. In fact, the instantiation of a parameterised collaboration originates another collaboration, but the instance of a design pattern is a design portion that should satisfy the requirements of the pattern and solve a concrete problem, it may even be purely structural and not include interaction.

A pattern should be used like a template to guide the designer in the flexible selection, creation and validation of the elements (classes, objects, relationships, attributes, methods, etc.,) that participate in the solution of a typical design problem. However, the "*ClassifierRole*" or "*AssociationRole*" of a parameterised collaboration depend on base classifiers or associations that should exist previously and where the "*ClassiferRole*" or "*AssociationRole*" are respectively a view of these. Also, they can only bind classifiers or associations to one parameter with the only restriction that the participant is of the same type (or a descendant of the type) as the corresponding parameter. On the other hand, it is impossible to specify patterns with a variable number of participants, the most frequent type, because the number of arguments should be similar to the number of parameters.

These and other inconveniences are an obstacle to be able to specify the essence of a pattern appropriately. For this reason we present the following approach.

4 Proposal for the modeling and treatment

In this work we propose a different model for the structural specification of design patterns and their integration with UML [6]. For this, we take advantage of the expressive capacity of a very well-known notation (UML) and we add a minimum group of modeling elements that allow us to define the invariant of a pattern establishing the appropriate mechanisms to check it.

This proposal arises from the idea that a pattern should help the designer in the flexible assignment, creation and validation of the different elements of modeling that participate in the solution.

Figure 1 shows a basic outline with the fundamental concepts that have been defined.

A *role-element* is the specification of a design element that will play a fundamental role in the solution proposed by a pattern. A role-element defines the type (class, object, attribute, operation, parameter, relationship, etc.) and the restrictions (structure and/or behaviour and/or relationships) that will satisfy this design element.

The role-elements are key pieces to model the restrictions that a pattern requires. These will be its essential components. A role-element uses the same symbol that the design element that is represented.

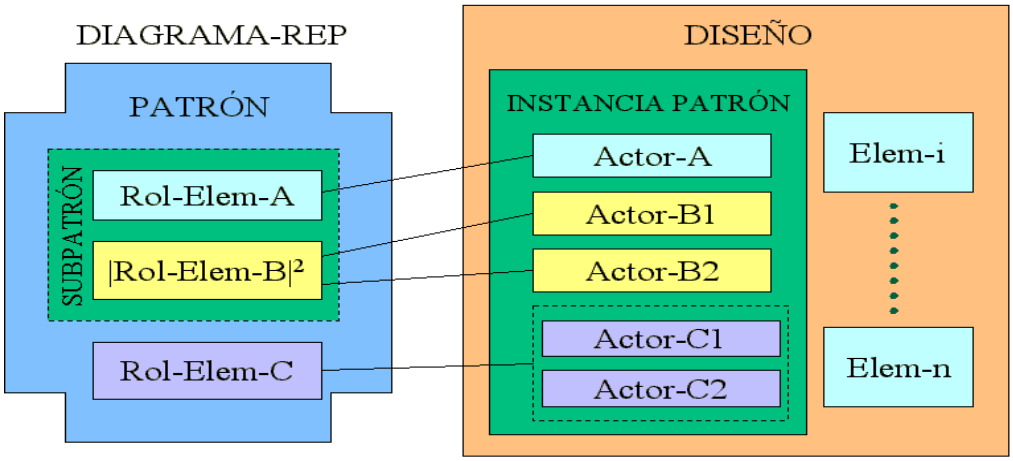

Figure 1. Basic concepts

A *REP diagram* (**R**ole-**E**lements of a **P**attern) is made up of the group of role-elements that define a certain pattern. The structure, behaviour and/or relationships that each role-element presents in this diagram represent the characteristics that the design elements of their instances should possess. Also, to complete the invariant of a pattern, other kinds of restrictions that should be completed by groups of participants can be specified (number of elements, organization of these, etc.). These restrictions can also be expressed in the REP diagram using a special notation.

A mechanism that connects design elements in a certain context with role-elements of a REP diagram could check the execution of the restrictions demanded for each element and could check if a group of design elements is an instance of the pattern.

When a design element completes all the restrictions that demands a certain role-element it can be an *actor* of this. In some ways one could think that it is assuming or playing the role of that role-element.

As has been said previously, the symbol associated to each role-element is the same as that of the actor or actors that will play it, however it is necessary to differentiate its semantics. The symbols of the actors represent elements that are part of the implementation and, consequently, their instances belong to the system in execution, however, the role-element represents design elements and it is their actors that can be instanciated.

Every role-element has an identity, that is to say, it should possess a unique identifier inside the REP diagram.

Two role-elements are different when the restrictions that they represent are different and must have different identifiers.

Two identifiers will be different if they show different identifiers or different format (italic, underlined, etc.).

Each type of design element will be characterized by some specific properties, and each type of role-element (role-class, role-attribute, role-relationship, role-operation, role-parameter, etc.) will define the restrictions on its possible actor. For example, a role-class will

define the role-relationships that it has with the other role-classes, it will indicate if its actor will be abstract, the role-attributes, the role-operations, etc.

A role-element can be:

- Obligatory. The role-element should have some actor.
- Optional. The role-element can have some actor.

A role-element can possess a *cardinal* that will indicate the number of actors that should play it.

The cardinal can be:

- Fixed. A concrete value will appear.
- Variable. Some variable will appear. The value of the cardinal is not known a priori. It can be any value or it can depend on the value of another cardinal variable if they share some of their variables. For example, if the cardinal of two role-elements is j the values will always coincide, if the cardinal is j in one place and 2*j in another the cardinal of the latter will be double the previous one.

If the cardinal is not specified it is fixed and its value is one.

A *bond* is the assignment of an actor to a role-element.

Often the instance of a pattern can present design elements that do not coincide exactly with some of the role-elements specified in the REP diagram of the pattern, disguising its ideal form. Sometimes this is because the restrictions of a role-element are satisfied by a group of elements. For example a role-relationship could be carried out by several relationships which transitively satisfy the restrictions imposed by this, the responsibilities of a class could be distributed among several classes, the behaviour required by a role-operation can be distributed among several methods, etc. Therefore, we denominate *simple bond* to that which is established between a role-element and a single actor that is forced to satisfy all the demanded restrictions and *shared bond* when it happens between a role-element and several actors that satisfy jointly all the required restrictions.

The following bond rules are applied:

- An actor can be linked to different role-elements.
- An actor cannot be linked twice to the same role-element.
- Several actors can be linked with the same role-element if their cardinal is greater than the unit or the bond is shared.
- A role-element can appear several times inside a REP diagram. In this case, the bond of an actor with one of its occurrences extends to the rest.
- The actors that participate in bonds with role-elements belonging to another role-element should be part of the actor bound to that other role-element (container).

A *bond* is *valid* when it completes the bond rules and the actor (simple bond) or actors (shared bond) complete the necessary requirements to be able to play the designated role-element. Failing this it will be *invalid*.

A *bond* is *complete* when at least all its obligatory role-elements are part of some valid bond and its cardinals are satisfied. Otherwise it will be a *partial bond*.

An *instance-pattern* is a diagram formed by the modeling elements that are actors of the role-elements of a certain REP diagram after a complete bond.

A *subpattern* is a defined subset of role-elements that belong to a REP diagram.

An *instance-subpattern* is a diagram formed by the modeling elements that are actors of a certain subpattern after a complete bond.

A subpattern can have a fixed or variable cardinal.

When a subpattern with a certain cardinal contains other cardinals belonging to subpatterns or internal role-elements we say that the cardinals are nested.

We have defined a simple visual notation (based on UML) that easily expresses all the notions mentioned previously.

As an example, we present the Abstract Factory pattern specified according to the notation used in Gamma et al. [5] and the associated REP diagram according to this new approach.

5 Application example. The Abstract Factory pattern.

This pattern (figure 2) provides an interface to create families of related objects without specifying their concrete classes.

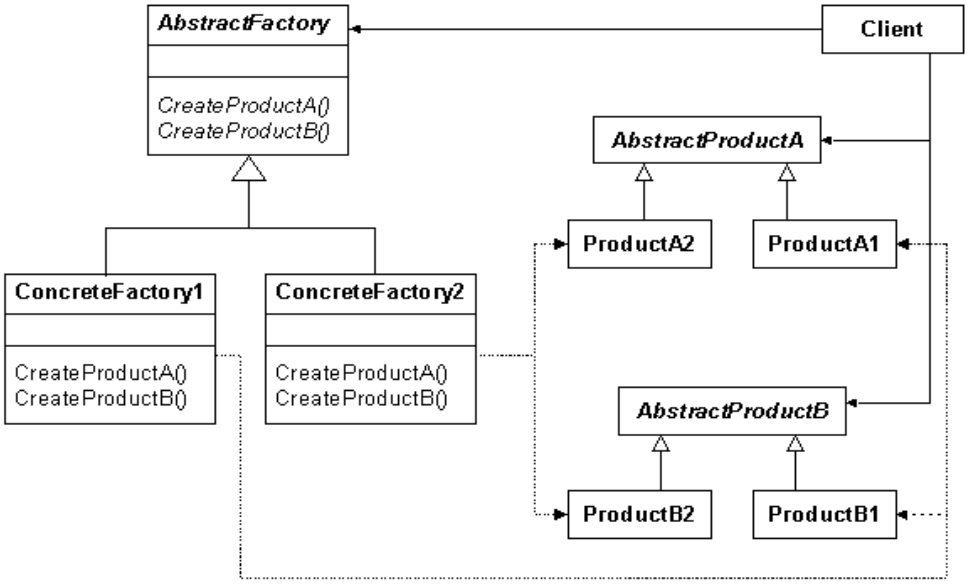

Figure 2. Abstract Factory pattern [5].

This specification is not general because it is based on a concrete example. In this case, one can observe that a variable number of classes or methods are specified by means of a fixed number of these, in this case two. This notation doesn't allow us to express the generality of a pattern without ambiguities.

In figure 3, we can observe that the REP diagram associated to this pattern is more general and simpler. The role-elements indicate the type and the structural restrictions which the actors must fulfill in order to be linked.

A restriction has been applied on the number of actors that will be able to play some of the role-elements using a variable cardinal. In this case, it is indicated that the role-element "*CreateProduct()*" abstract, "CreateProduct()" concrete and the subpattern can have a variable number of actors, but it should be the same one for all because the variable "i" is shared. This situation is also presented in "ConcreteFactory" and "Product" where the variable "j" is shared.

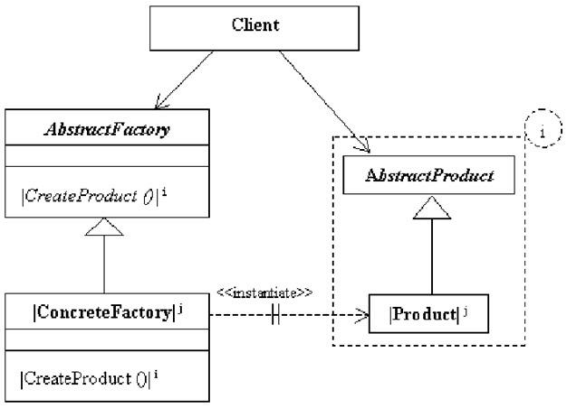

Figure 3. REP Diagram corresponding to the Abstract Factory pattern.

It is also possible to observe a restriction on the role-relationship that exists between "ConcreteFactory" and "Product". This restriction is represented by two parallel lines that cut the line of the role-relationship perpendicularly and expresses that this relationship is biyective, that is to say, the relationships between the groups of actors of the role-element origin and destination should form an biyective application.

Figure 4 presents an instance of the pattern. We can check the restrictions that the pattern imposes. By means of a color that identifies the pattern the role-element is visualized. If the element participates in different patterns, these can be specified sequentially. This characteristic could be visualized or hidden as suits by means of a tool.

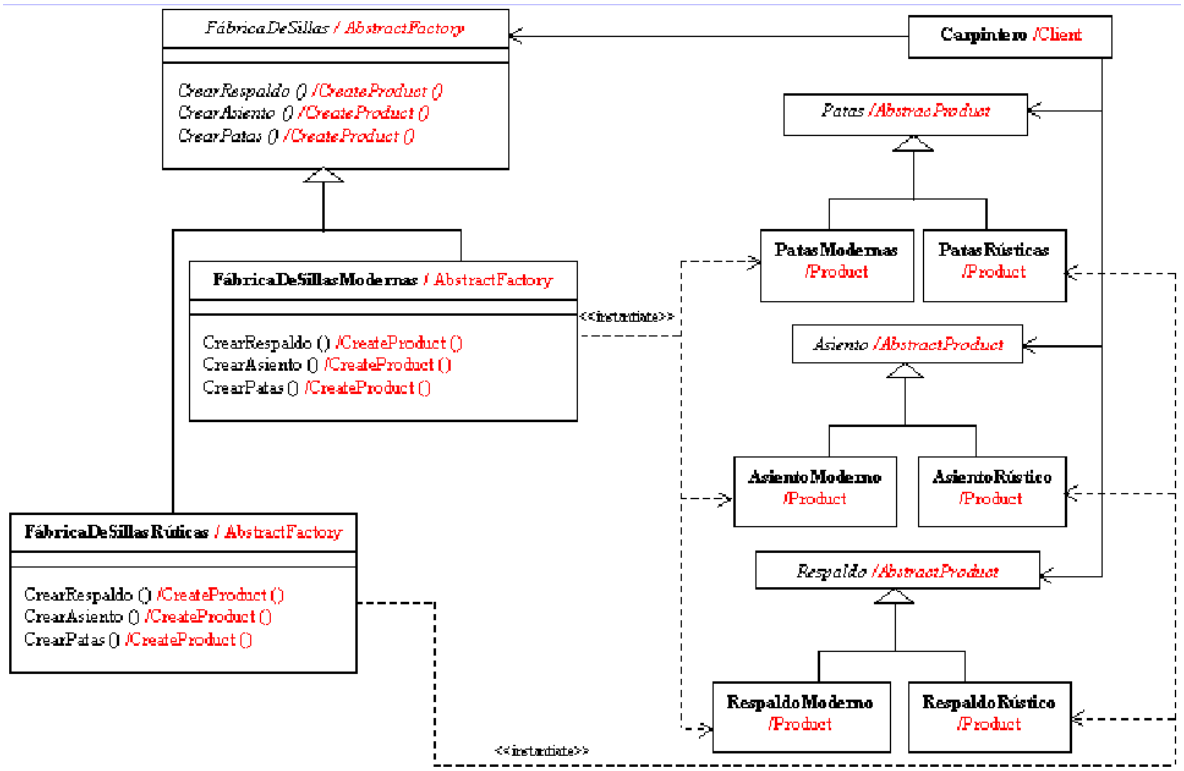

Figure 4. An instance of the REP diagram corresponding to the Abstract Factory pattern.

5 Conclusions and future work

To carry out the precise specification of a design pattern is complicated because it is not easy to give answers to the following key questions:

- What aspects should be specified?, that is to say, what should be part of the essence of a pattern?. Must not be forgotten that a pattern describes a problem, a solution and the consequences of its use. Being aware that the intrinsic complexity of a pattern is great, it seems to be that the existing works, including the present work, they are centered mainly on the more or less precise specification of that solution. At the moment, in this work we have centered on the structural aspects of this solution, however, is left pending for future works the specification of the behaviour properties.
- How to carry out this specification?. The answer necessarily depends on the answer given to the previous question. The pattern here presented carries out a precise and visual specification of the structure of a pattern through a simple extension to UML, without necessity of appealing to a formal notation as OCL. The result is a familiar notation to the designer, simple, complete and easy to learn, and an intuitive model that treats the design patterns as if they were true templates.

At the moment we have been able only to specify the structural restrictions in a visual and precise way, by means of REP diagrams. Regarding the specification of the behaviour properties, we think that it is necessary to investigate how to adapt the interaction diagrams of UML to the abstraction level of a pattern, how an extension of OCL with temporal logic could help, etc.

We think that the characteristics of the specification model that we have presented can be the base for the development of a future tool of great interest for the designers.

6 References

[1] Eden, A. H. (2000). “Precise Specification of Design Patterns and Tool Support in their Application”, PhD diss., Department of Computer Science, Tel Aviv University.

[2] Eden, A. H. (2002). “LePUS: A Visual Formalism for Object-Oriented Architectures”. The 6th World Conference on Integrated Design and Process Technology. Pasadena, California, June 26-30, 2002.

[3] Eden, A. H., Gil, J. and Yehudai, A. (1997). “Precise Specification and Automatic Application of Design Patterns”. In The 12th IEEE International Automated Software Engineering Conference – ASE 1997.http://www.math.tau.ac.il/~eden/bibliography.html#ase

[4] Eden, A. H., Hirshfeld, Y. and Yehudai, A. (1998) “LePUS – A Declarative Pattern Specification Language”. Technical report 326/98, department of computer science, Tel Aviv University. 1998. http://www.math.tau.ac.il/~eden/bibliography.html#lepus

[5] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). Design Patterns: Elements Reusable of Object-Oriented Software. Addison Wesley Professional Computing Series, Reading, MA.

[6] Isla, J.L. (2002). “A new approach for modelling design patterns with UML”. The 12th PHDOOS workshop (ECOOP2002), Málaga, 10-14 june 2002, Spain. http://www.softlab.ece.ntua.gr/ facilities/public/AD/phdoos02/jose.ps

[7] Kent, S. (1997). “Constraint Diagrams: Visualising Invariants in Object-Oriented Models”. In Proceedings of OOPSLA’97. ACM Press.

[8] Lauder, A. and Kent, S. (1998). “Precise Visual Specification of Design Patterns”. In Proceedings of ECOOP’98. Springer-Verlag.

[9] Le Guennec, A., Sunyé, G. and Jézéquel, J. (2000). “Precise Modeling of Design Patterns”. In Proc. UML’ 2000. Springer Verlag, LNCS 1939.

[10] Meijers, M. (1996). Tool Support for Object-Oriented Design Patterns. Master’s Thesis, INF-SCR-96-28. Utrecht University.

[11] Object Management Group. (2002). OMG Unified Modeling Language Specification V 1.4., http://www.omg.org

[12] Riehle, D. (1996). “Describing and Composing Patterns Using Role Diagrams.” WOON ’96 (1st International Conference on Object-Orientation in Russia), Conference Proceedings. St. Petersburg Electrotechnical University, 1996. Reprinted in K.-U. Mätzel and Hans-Peter Frei (eds.), Proceedings of The Ubilab Conference ’96, Zürich. Germany, Universitätsverlag Konstanz, pp. 137-152.

[13] Sunyé, G., Le Guennec, A. and Jézéquel, J. (2000). “Design Patterns Application in UML”. In Proc. European Conference in Object Oriented Programming – ECOOP’ 2000. Springer Verlag, LNCS 1850.