IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4700

A Metamodel for Specifying Design Patterns in UML

Reza Jaberi and Mohammadreza Razzazi
Software Research and Development Laboratory
Computer Engineering Department
Amri-Kabir University of Technology
jaberi@b-et.com, razzazi@ce.aut.ac.ir

ABSTRACT

UML considers a pattern as a parameterized collaboration between objects replaced by their actual values during the pattern application phase. However, a design pattern differs from a simple design construct described by the collaboration mechanism. A pattern does not express a fixed and accurate structure. Rather, it defines a generic template whose different aspects can be changed in different applications. In this paper, UML is extended to provide the linguistic infrastructure that specifies design patterns. Most of the existing pattern specification approaches consider a specific solution of the pattern problem to describe the pattern. However, a pattern should be considered as a family of solutions and applications of the pattern problem; since, an unbounded number of concrete solutions and real applications may conform to it. In this paper, four models are presented to describe the general specifications of a pattern, its essence, its solutions and its instances, respectively. The main construct used in these models is the role concept that defines a family of classes. This paper also proposes a model to specify the definitions related to this concept. This model uses three relations to define the binding between the roles specification level and their implementation level. These relations define the implementation of roles, satisfying the structural relationships between roles and conforming to the behavioral specifications of roles, respectively. Finally, the semantics of the proposed models are precisely defined. This paper ends up presenting the definitions needed to support the pattern-oriented features including the pattern specification, application, validation, recognition and discovery.

1. INTRODUCTION

Design patterns are considered as an approach to encapsulate design experiences in the software development. Each pattern expresses a relation between a certain context, a problem and a solution [1]. According to the object-oriented software design, it can be viewed as a description of communicating objects and classes that are customized to solve a general design problem in a particular context [2]. Design patterns are mostly described in the form of a specific solution using object-oriented diagrams.

The current version of the Unified Modeling Language [3] has defined a mechanism to support design patterns. This mechanism considers a pattern as a parameterized collaboration whose parameters are replaced by their actual values during the pattern application phase. Two conceptual levels are provided in this mechanism. Collaboration diagrams are used to describe design patterns and the collaboration usage notation is used to represent their applications.

The UML proposed mechanism has many problems in specifying design patterns that some of them are discussed in [5]. Most of these problems originate from the fact that UML has not considered the design pattern as a distinct language construct. However, there is a point that differentiates a design pattern from a simple design construct described by the collaboration mechanism. A design pattern does not express a fixed and accurate structure. Rather, it defines a generic template whose different aspects can be changed in different contexts.

Another problem is that collaboration diagrams are used as the tools of the behavioral modeling of a system. Restricting pattern specifications to collaborations generally means that the modeling of a system does not need any structural diagrams and behavioral diagrams are sufficient. Moreover, most of the known pattern specification templates, such as the one presented in the catalog of design patterns [2], consider distinct sections for the structural and behavioral features.

Our goal in this paper is to extend UML to provide the linguistic infrastructure for specifying design patterns. Most of the existing pattern specification approaches, including the one proposed by UML, have a fundamental weakness in their view. They consider a specific solution of the pattern problem to describe the pattern. Nevertheless, a pattern is not restricted to a particular solution; rather, unbounded number of concrete solutions and real applications may conform to it. It is better to consider a pattern as a family of solutions and applications of the pattern problem. This view is compatible with the definition of patterns in [6].

The mentioned view leads to a 3-layer structure presented in [7] to describe design patterns. The first layer describes the essence of a pattern. A family of solutions implements this essence in the second layer. A family of real applications in the third layer applies each of these solutions. Based on this structure, we present four models that respectively describe the general specifications of a pattern, its essence, its solutions and its applications. We also introduce a mechanism to represent design patterns based on this approach.

The main construct used in these models is the *role* concept, which has been introduced by the object-oriented methods such as OORam [9]. While a class is a complete description of objects, a role is a specific view a client object holds on the other object. In this paper, we propose a model that specifies the required definitions related to this concept. We define the binding between the roles specification level and their implementation level by three relations. These relations define the implementation of roles, satisfying the structural relationships between roles and conforming to the behavioral specifications of roles, respectively.

Finally, we precisely define the semantics of the proposed models and present the required definitions in order to the pattern-oriented features. These features according to [8] include the pattern specification, application, validation, recognition and discovery.

2. UML PROPOSED MECHANISM

In order to support design patterns, UML defines two conceptual levels that are discussed in this section.

2.1. Collaborations in UML

A collaboration is a description of a collection of objects that interact to implement some behavior within a context [4]. A collaboration is defined in terms of *roles* whose definitions has correspondence with the concepts discussed in the object-oriented methods such as OOram [9]. A role is a specific view a client object holds on the other object. It is the client that defines what constitutes the role [10]. An object can play different roles at different times, and may also play more than one role at the same time [11].

Roles are specified using *ClassifierRoles* and *AssociationRoles* in UML. A ClassifierRole represents a description of the objects that can participate in an execution of the collaboration. ClassifierRoles are connected to each other by means of AssociationRoles. A ClassifierRole does not specify the complete features of an object; it only specifies the features required to play that role. An object conforms to a ClassifierRole if it provides the required features of the role. Thus, each ClassifierRole is a distinct usage or a restriction of some

Classifiers in the context of a collaboration. These Classifiers are considered as the *base* of that ClassifierRole. Instances of the base Classifiers or their descendants can play that role.

2.2. Specifying Design Patterns

UML considers a design pattern as a parameterized collaboration between objects. A parameterized collaboration represents a design construct that can be reused in various designs. The structural and behavioral aspects of a pattern solution are modeled as a collaboration and the elements that must be bound to the elements in a particular context are considered as its parameters. According to the definitions of the collaboration elements in UML, the base classifiers of each role are turned into parameters of the collaboration.

2.3. Applying Design Patterns

A design pattern is instantiated by supplying the actual classifiers for the parameters in the pattern definition. Each instantiation yields a collaboration among a specific set of classifiers in the model. The instantiation of a pattern is showed by means of the collaboration usage notation. This notation is a dashed ellipse containing the name of the pattern. A dashed line is drawn from this symbol to each of the classifiers participating in the pattern and each line is labeled with the role of the participant.

2.4. Existing Problems

The UML proposed mechanism to describe design patterns has many problems that some of them are mentioned here and other ones are discussed in [5].

An important point is that a collaboration defines the features needed for an object to participate in an interaction with some other objects and specifies how the interaction is done. Parameterized collaborations are as well used to produce collaborations in different contexts by assigning different values to the template parameters. Thus, these collaborations do not define any new structural specifications. Restricting pattern specifications to collaborations generally means that the modeling of a system does not need any structural diagrams and behavioral diagrams are sufficient. It is quite clear that it is not true and structural diagrams, such as class diagrams, are required to model a system. Most of the known pattern specification templates, such as the one presented in the catalog of design patterns [2], allocate distinct sections to structural and behavioral features.

At last, most of the existing pattern specification approaches, including the one proposed by UML, have a fundamental weakness in their view. They consider a specific solution of the pattern problem to describe the pattern and provide concrete diagrams intended to present the specific solution. However, a pattern is not restricted to a particular solution and an unbounded number of concrete solutions and real applications may conform to it. Patterns community explicitly introduced the *"used twice rule"* as follows: A pattern is a pattern, if and only if it has been used in more than one application [12]. Thus, a pattern is never the result of a single analysis, but only of a cross-project analysis. It is better to consider a pattern as a family of solutions and implementations of the pattern problem, which is compatible with the definition of patterns in [6].

3. SELECTED STRUCTURE

This section illustrates the structure used to describe design patterns based on the mentioned view.

3.1. Main Construct

A participant of a pattern defines the features required in the pattern context. According to the definitions of the role concept discussed in the section 2.1, it seems that the participants of a pattern description are more compatible with roles rather than classes. In fact, describing design patterns requires a design construct that specifies a family of classes and the role concept is more appropriate for this purpose.

Gamma et. al proposed the *"program to an interface, not an implementation"* principle in their catalog as follows: "Don't declare variables to be instances of particular concrete classes. Instead, commit only to an interface defined by an abstract class."[2] They note that this principle is a common theme of the design patterns they describe. This principle is very close to the role concept. Pattern community has also noted that roles are the true structural primitives upon which most patterns rely [13] and classes are only used because a role modeling primitive is not yet established.

3.2. Family of Solutions and Applications

The 3-layer structure presented in [7] has been selected to describe a design pattern as a family of solutions and real applications. Role and class diagrams are used at different levels of abstraction in this structure. Role diagrams are specification-oriented, while class diagrams are implementation-oriented. Both are needed, since a good design pattern should comprise a clear problem solution as well as efficient ways of implementing it.

The first layer describes the essential properties of a design pattern without any additional property and is common between all of its solutions. This common core for all possible variants of a design pattern is called its *essence* in [14]. Different variations of a pattern are special instances of its essence. Role diagrams are used to present the essence of a pattern, since they do not prescribe a certain class design, but only set up constraints in such a way that the actual core of the solution is maintained.

A pattern specified by a role diagram can be made more concrete by a class template, which in turn can be instantiated in numerous applications. Thus, a role pattern specifies and abstracts from a family of class patterns, which in turn specify and abstract from a family of concrete implementations. The third level has another interpretation and based on the template presented in [2], it can be considered as the samples of the pattern.

4. PROPOSED ROLE MODEL

The current specification of the role concept in UML has some problems that are mentioned in this section and a model is proposed to solve them.

4.1. Current Problems

The most important problems of the UML role support are the following three:

- *Independent Definition*: The UML specification does not allow a classifier role to exist with no base classifier. Nevertheless, there are many cases, such as the RUP method [15] and the design pattern specifications, in which roles are defined before classes and then classes are introduced to implement them. So, roles should be defined independent of base classifiers.
- *Base Classifier*: A classifier role has no feature from its own. It defines a subset of those available in the base. In fact, roles in UML are mappings between collaborations and their instances. However, implementing roles is wider than this simple mapping idea and the references such as [16] and [18] discuss different views in this area. So, it is better to define roles with their own features and force classes to implement them.
- *Association between Roles*: An association role is a specific usage of at most one base association and is defined dependent on that. However, it is possible that some different and general associations satisfy the conditions stated in that usage. So, association roles should be independently defined and satisfying them should be considered a distinct concept as the implementation of roles.

4.2. Revised Generalization Model

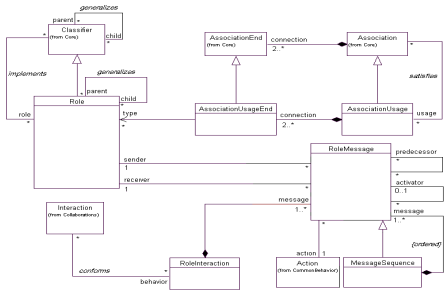
The UML specification defines the inheritance relationship using an instantiable metaclass named *Generalization* [3]. In order to get all of the parents of a given element, all of the Generalization instances must be found in which the mentioned element is defined as the specialized one. Such a question is much needed in our proposed models. Accordingly, another model is taken from [17] and [19] to define the inheritance relation. This model defines a new relation named *generalizes* by which a class or a role specifies the classes or the roles inherited from.

4.3. The *RoleModel* Package

A package named *RoleModel* is introduced to solve the mentioned problems. It is shown in figure 1 and is the linguistic infrastructure that specifies the role concept, the structural relationships between roles and the behavioral specifications of roles.

A new metaclass named *Role* is defined to specify roles. Each role has its own features and can be defined independent of its implementing classes. The *implements* relation specifies the relationship between a role and the imple-

Figure 1: *The structure of the* RoleModel *package*



menting classes. This relation relates classifiers to the roles that can be played by their instances.

The structural relationships between roles are defined using associations between them. An association role is specified independent of its base by means of the *AssociationUsage* metaclass. The scope of an association is wider than that of its usage in an association role. A new relation, named *satisfies*, relates an association to its specific usage.

Each communication between two roles is done by means of sending a message between them and is specified by the *RoleMessage* metaclass. An interaction between a set of roles involves a set of communications among them to carry out a particular purpose and is specified by the *RoleInteraction* metaclass. It imposes some behavioral constraints on the implementing classes. Its implementation by an actual interaction between the implementing classes is specified using a new relation named *conforms*.

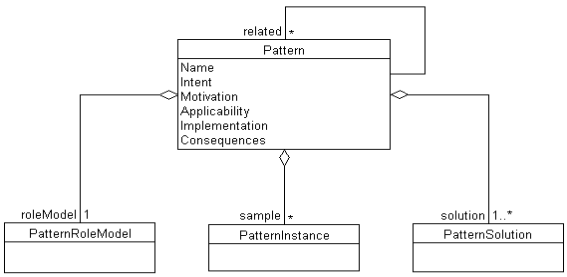
5. PATTERNMODEL PACKAGE

This section extends the UML metamodel to provide the linguistic infrastructure that specifies design patterns based on the approach discussed in the previous sections. The proposed models are contained in a package named *PatternModel*.

5.1. Pattern Model

Figure 2 shows the model suggested to describe the specifications of design patterns. It is based on the features presented in the catalog of design patterns [2]. The features such as the role model, solution, samples and related patterns are determined by the relationships between the model elements. The features such as the name, intent, motivation, applicability and consequences are considered as the attributes of the pattern.

Figure 2: *Pattern model*

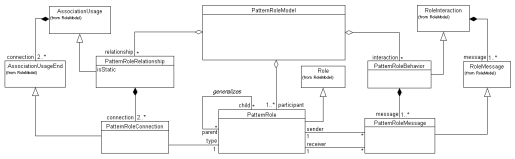


5.2. Pattern Role Model

Each pattern has a role model that defines the pattern essence and figure 3 shows the required model to describe it. A pattern role model is specified by the *PatternRoleModel* metaclass and involves some roles specified by the *PatternRole* metaclass. These roles must be implemented by the classes participating in the pattern solution model.

Pattern roles are connected to each other using some relationships specified by the *PatternRoleRelationship* metaclass. Some of these relationships indicate the static features of roles and some of them exist only in the context of the pattern to provide a behavioral communication path.

Figure 3: *Pattern role model*



Pattern roles communicate with each other by sending messages specified by the *PatternRoleMessage* metaclass. A pattern role model contains some interactions specified by the *PatternRoleBehavior*, which define the sequences of messages and the control flow of the role model.

5.3. Pattern Solution Model

Each pattern has some solution models that implement the pattern role model and figure 4 shows the model required for describing them. A pattern solution model is specified by the *PatternSolution* metaclass and involves some participants specified by the *PatternClass* metaclass. As mentioned in the section 3, these participants are not true classes, they are roles that implement the roles defined in the pattern role model. They are called classes because their tendency is towards the implementation.

The behavioral features of a pattern class are of two kinds named *Template* and *Hook* methods in [20]. They relate to some pattern core functionality and application specific functionality, respectively. Template methods have specific implementations and provide a general skeleton for the pattern behavior. The functionality of this skeleton is changed in particular contexts by hook methods. Hook methods are called in the implementation of template methods and are defined when the pattern is applied.

Pattern classes are related to each other using some relationships specified by the *PatternRelationship* metaclass. The establishment of these relationships when applying the pattern depends on how its connections become visible and includes these kinds: association relationship, method parameter, local scope, global scope and the class itself. The first type is a static relationship in the application model and the others are transient relationships that provide a path for sending message. Accordingly, the static relationships of the pattern role model must be satisfied by an association type relationship in the solution model; however, the other relationships of the role model can be satisfied using each type of the relationships in the solution model.

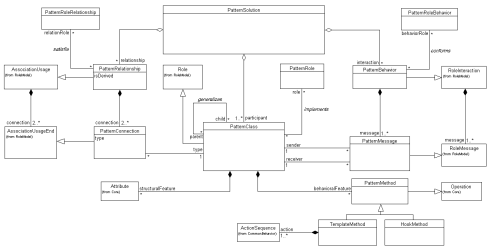
The behavior of a pattern solution is described by means of some interactions specified by the *PatternBehavior* metaclass. Each interaction involves some messages specified by the *PatternMessage* metaclass that are passed between the pattern classes to carry out a specific pattern purpose. Each of these interactions must conform to one of the interactions in the pattern role model.

There is a difference between the structural and the behavioral constraints on a solution. The solution structure must satisfy all of the structural constraints somehow, but all of the solution behaviors must conform to one of the behavioral constraints. The structure of a role model makes it possible to take place different behaviors and the behavioral specifications specify the valid ones between them. The pattern solution must present some valid behaviors and not necessarily all the valid behaviors.

5.4. Pattern Instance Model

A pattern is instantiated in a particular application by implementing one of its solutions. Figure 5 shows the required model to describe the instantiation

Figure 4: *Pattern solution model*



of a pattern in an application model. A pattern instance is specified by the *PatternInstance* metaclass. An application model involves some *classifiers* that implement pattern classes.

Application classifiers are related to each other by means of some *associations* that can be static or dynamic. The static ones are relationships that exist outside the pattern context too, and satisfy the pattern relationships whose types are specified as association in the pattern solution model. The others are transient relationships that are only established in the pattern context and can be used to satisfy the other kinds of the pattern relationships.

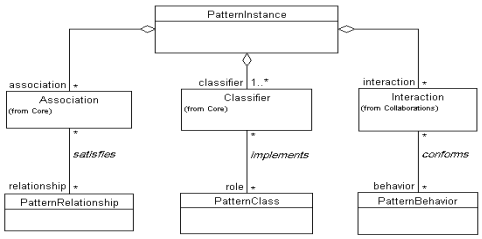
The behavior of the instances of application classifiers is specified by some *interactions* between them. Each interaction must conform to one of the pattern solution behaviors.

Further details discussed in the pattern solution model about implementing roles, satisfying structural relationships and behavioral conformance are also applicable here and we avoid repeating them.

6. SEMANTIC DEFINITION

This section defines the semantics of the proposed models. At first, the conditions required to implement a role model are expressed and at the end of this section, precise definitions are presented for instantiating a design pattern.

Figure 5: *Pattern instance model*



6.1. Implementing Roles

Definition 1: Implementing a Role
A class *C implements* a role *R*, if and only if
? *allAttributes(R)* $\subseteq$ *allAttributes(C)*
? *allOperations(R)* $\subseteq$ *allOpertions(C)*
? *R.parent* $\subseteq$ *(C.role* $\cup$ *C.parent.role)*

The attributes and the operations of the role must be a subset of those of the class. The class must also be able to implement the generalized roles of the role.

Definition 2: Implementing a Set of Roles
A set of classes *CS implements* a set of roles *RS*, if and only if
- $\forall$ *R* $\in$ *RS*, $\exists$ *C* $\in$ *CS where C implements R*

6.2. Satisfying Association Roles

Definition 3: Satisfying a Role Connection
A connection *CC* to an association between classes *satisfies* a connection *RC* to an association between roles, if and only if
- *CC.type implements RC.type*
  OR
  $\exists$ *C* $\in$ *CC.type.child where C implements RC.type*
- *RC.isNavigable* $\Rightarrow$ *CC.isNavigable*
- *RC.cardinality* $\subseteq$ *CC.cardinality*

The definition says that the corresponding class of the class connection or one of its subclasses must implement the corresponding role of the role connection and the navigability and cardinality constraints of the association between classes must be saved in its specific usage in the role model.

Definition 4: Satisfying a Set of Role Connections
A set of connections *CCS* to an association between classes *satisfies* a set of connections *RCS* to an association between roles, if and only if
? $\forall$ *RC* $\in$ *RCS*, $\exists$? *CC* $\in$ *CCS where CC satisfies RC*
? $\forall$ *CC* $\in$ *CCS*, $\exists$? *RC* $\in$ *RCS where CC satisfies RC*

Definition 5: Satisfying an Association Role
An association between classes *CR satisfies* an association between roles *RR*, if and only if
- *CR.connection satisfies RR.connection*

6.3. Conforming to Behavior of Roles

Definition 6: Conforming to a Message
A message *CM* from a behavioral model of classes *conforms* to a message *RM* from a behavioral model of roles, if and only if
? $\exists$ *CR* $\in$ *allAssociations(CM.sender)* $\cap$ *allAssociations(CM.receiver),*
$\exists$ *RR* $\in$ *allAssociations(RM.sender)* $\cap$ *allAssociations(RM.receiver) where CR satisfies RR*
? *CM.sender implements RM.sender*
? *CM.receiver implements RM.receiver*
? *CM.action = RM.action*

The definition indicates that the sender and the receiver must provide the path required to send the message. They must also implement the sender and the receiver roles, respectively. The executed action upon receiving the message must be the same.

Definition 7: Conforming to a Message Sequence
A sequence of messages $CM_1, CM_2, ..., CM_n$ sent in a behavioral model of classes *conforms* to a sequence of messages $RM_1, RM_2, ..., RM_m$ sent in a behavioral model of roles, if and only if
? *n = m*
? $\forall i \in \{1..n\}$, CM_i *conforms* RM_i

Definition 8: Conforming to an Interaction Role
An interaction *CI* from a behavioral model of classes *conforms* to an interaction *RI* from a behavioral model of roles, if and only if
- $\exists$ *MS* $\subseteq$ *CI.message where MS conforms RI.message*

The definition says that a subsequence of the class messages must conform to the sequence of the role messages. A role model indicates a general behavior and a high-level control flow that can be more complex in an actual application.

6.4. Implementing a Role Model

Before defining the implementation of a role model, the following definitions are considered for class models and role models:

Definition 9: Class Model
A set of classes *CS* along with their structural relationships in a set *CRS* and their behavioral specifications in a set *CIS* is considered as a class model represented as *<CS, CRS, CIS>*. Class model is also known as application model.

Definition 10: Role Model
A set of roles *RS* along with their structural relationships in a set *RRS* and their behavioral specifications in a set *RIS* is considered as a role model represented as *<RS, RRS, RIS>*.

Now, the implementation of a role mode is defined as below:

Definition 11: Implementing a Role Model
A class model *CM<CS, CRS, CIS> implements* a role model *RM<RS, RRS, RIS>*, if and only if
? *CS implements RS*
? $\forall$ *RR* $\in$ *RRS*, $\exists$ *CR* $\in$ *CRS, CR satisfies RR*
? $\forall$ *CI* $\in$ *CIS*, $\exists$ *RI* $\in$ *RIS, CI conforms RI*
In this case, the class model can be considered as an *instance* of the role model.

Namely, the classes of the class model must implement the roles of the role model, each relationship of the role model must be satisfied by one of the relationships of the class model and each interaction of the class model must conform to one of the interactions of the role model.

6.5. Instantiating a Design Pattern

Definition 12: Pattern Solution
A pattern solution model *PS* is a solution of a design pattern *P*, if and only if
? *<PS.participant, PS.relationship.nonTransient, PS.interaction> implements <P.roleModel.participant, P.roleModel.relationship.static, P.roleModel.interaction>*

This definition indicates that the given solution model must implement the pattern role model.

Definition 13: Pattern Instance
A pattern instance model *PI* is an instance of a design pattern *P*, if and only if
? $\exists PS \in P.solution$ *where*
<*PI.classifier, PI.association.nonTransient, PI.interaction*> *implements*
<*PS. Participant, PS.relationship.nonTransient, PS.interaction*>

This definition indicates that the given instance model must implement one of the solutions of the design pattern. The above definitions implicitly consider the non-static relationships of the pattern role model and the transient relationships of the pattern solution model as one of the message conformance conditions during the behavioral checking of the solution and the instance, respectively.

7. PATTERN-ORIENTED FEATURES

This section presents the required definitions in order to support pattern-oriented features. It uses the definition of a pattern instance given in the previous section. According to [8], pattern-oriented features include specification, application, validation, recognition and discovery.

Definition 14: Pattern Specification
A design pattern is s*pecified* by a pattern model *P*, a pattern role model *RM*, a set of pattern solutions *PSS*, a set of pattern instances *PIS* and a set of related patterns *RPS*. It is represented as <*P, RM, PSS, PIS, RPS*>.

Definition 15: Pattern Application (Instantiation)
Given a design pattern *P*, an *application* generates or modifies a pattern instance *PI* such that *PI* is an instance of *P* based on definition 13.

Definition 16: Pattern Validation
Given a design pattern *P* and a pattern instance *PI*, a *validation* answers the question whether *PI* is an instance of *P* based on definition 13.

Definition 17: Pattern Recognition
Given a design pattern *P* and an application model *M*, a *recognition* gives all pattern instances *PI* in *M* which are instances of *P* based on definition 13.

Definition 18: Pattern Discovery
Given a set of application models *MS* and a set of known patterns *PS*, a *discovery* is a search for pattern instances *S*<*P, RM, PSS, PIS*> such that
? $S \notin ? PS$
? *PIS* is a set of submodels of *MS* having at least two members and contains the instances of *P* based on definition 13.

The last definition is based on the "*used twice rule*" that indicates a pattern must be used in more than one application.

8. CONCLUSION

In this section, the proposed models are evaluated and the benefits of the presented approach regarding the UML proposed mechanism are mentioned.

1. As mentioned, collaborations are behavioral modeling tools. Accordingly, the structural specifications of design patterns are not explicitly considered in UML. The proposed role and pattern models consider the structural specifications as distinct parts.
2. A generalization relationship in a collaboration does not necessarily force such a relationship between the base classifiers. It only indicates that the child role inherits the parent features. The proposed role and pattern models explicitly consider the generalization between participants.
3. UML assumes that the bases of association roles can be automatically deduced from the existing associations among the corresponding base classifiers when the pattern is bound. The proposed role model considers satisfying association roles as a distinct concept. The proposed pattern model defines different kinds of satisfying a pattern relationship. It also considers the usage of derived associations for satisfy a relationship.
4. Classifier roles are not allowed in UML to have any features of their own. They only repeat some parts of the features of their bases. The proposed role model drops this restriction and gets rid of the base altogether. Roles are allowed to define the needed features within themselves.
5. UML does not define how the pattern interactions are involved in the binding process. The proposed role and pattern models consider the interactions as a new kind of constraint. The actual participants must conform to these behavioral constraints.
6. In the UML proposed mechanism, the existing elements of the application are mapped on the elements of the pattern. Applying design patterns is more than this simple mapping idea and may generate or modify some elements. The proposed role model defines a role as a general description for a family of classes and there is no need to a preexisting structure.
7. The essence of a pattern can not be described by the UML proposed mechanism. The proposed pattern model considers a distinct level for the pattern essence and describes it using role diagrams.
8. UML does not support pattern-oriented features appropriately. It has only considered pattern specification and application. Pattern-oriented features, including specification, application, validation, recognition and discovery are well supported in this paper.

REFERENCES

[1] Christopher Alexander, *The Timeless Way of Buildin*g, Oxford University Press, New York, 1979.

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, 1995.

[3] UML Revision Task Force, *OMG Unified Modeling Language Specification v. 1.*3, Document ad/99-06-08, Object Management Group, June 1999.

[4] Grady Booch, Ivar Jacobson, James Rumbaugh, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[5] Gerson Sunyé, Alain Le Guennec, Jean-Marc Jézéquel, "Design Patterns Application in UML", ECOOP'2000, *LNCS 1850*, pp. 44-62, 2000.

[6] Ralph Johnson, "An Introduction to Patterns", *Report on Object Analysis and Design*, vol. 1, no.1, SIGS Publications, 1994.

[7] Dirk Riehle, "Describing and Composing Patterns Using Role Diagrams", *Proceedings of the 1996 Ubilab Conference*, pp. 137-152, 1996.

[8] Amnon Eden, Yoram Hirshfeld, "Towards a Formal Foundation for Object Oriented Architecture", *Proceedings The 3rd ACM SIGSOFT Workshop on Formal Methods in Software Practice*, 2000.

[9] T. Reenskaug, P. Wold, O. Lehne, *Working with Objects*, Greenwich, Manning, 1996.

[10] B. Kristensen, K. Osterbye, "Roles: Conceptual Abstraction Theory and Practical Language Issues", *Proceedings of Theory and Practice of Object Systems*, 1996.

[11] Barbara Pernici, "Objects with Roles", *Proceedings ACM-IEEE Conference of Office Information Systems (COIS)*, 1990.

[12] J. Vlissides, M. Linton, "Unidraw: A Framework for Building Domain-Specific Graphical Editors", *ACM Transactions on Information Systems* 8, 3, pp. 237-268, 1990.

[13] Frank Buschmann, "Falsche Annahmen (Teil 2)", *OBJEKTspektrum*, pp. 84–85, 1998.

[14] Amnon Eden, "Giving "The Quality" a Name", *Journal of Object Oriented Programmin*g, Guest column, SIGS Publications, 1998.

[15] Grady Booch, Ivar Jacobson, James Rumbaugh, *The Unified Software Development Process*, Addision-Wesley, 1999.

[16] Friedrich Steimann, "On the Representation of Roles in Object-Oriented and Conceptual Modeling", *Proceedings of* Data & *Knowledge Engineering 35*, pp. 83-106, 2000.

[17] Friedrich Steimann, "A Radical Revision of UML's Role Concept", *Proceedings of «UML» 2000 - The Unified Modeling Language*, Third International Conference, 2000.

[18] Martin Fowler, "Dealing with Roles", PLoP'97, *Conference Proceedings*, 1997.

[19] Bernd-Uwe Pagel, Mario Winter, "Toward Pattern-Based Tools", PLoP'96, *Conference Proceedings*, 1996.

[20] Wolfgang Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, Reading, 1995.