JOURNAL OF OBJECT TECHNOLOGY
Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2002
Vol. 1, No. 5, November-December 2002

UML Extensions for Design Pattern Compositions

Jing Dong
Department of Computer Science, University of Texas at Dallas, Texas, USA

Design patterns document good solutions to recurring problems in a particular context. Composing design patterns may achieve higher level of reuse by solving a set of problems. Design patterns and their compositions are usually modeled by UML diagrams. When a design pattern is applied or composed with other patterns, the pattern-related information may be lost because traditional UML diagrams do not track this information. Thus, it is hard for a designer to identify a design pattern when it is applied or composed. In this paper, we present notations to explicitly represent each pattern in the applications and compositions of design patterns. The notations allow us to maintain pattern-related information. Thus, a design pattern is identifiable and traceable from its application and composition with others.

1 INTRODUCTION

Design patterns [9] capture the distilled experience of expert designers. A design pattern systematically names, explains, and evaluates important and recurring design. The composition of design patterns [16, 10, 1, 5, 4] enable a higher level of reuse than individual design patterns and objects. The modeling and representation of design patterns and their compositions are usually based on object-oriented modeling techniques that use graphical notations such as the Unified Modeling Language (UML) [3, 14]. UML is a general-purpose language for specifying, constructing, visualizing, and documenting artifacts of software-intensive systems. It provides a collection of notations to capture different aspects of the system under development.

Notations are important for conveying concepts and information. Most notations can make complex concepts easy to understand and grasp. Nevertheless, some notations may cause misunderstandings and confusions. Good notations and correct uses of them can result in significant gains in terms of precision, expressiveness, unambiguity, succinctness, simplicity, clarity and compactness [15].

There are several different kinds of notations for modeling object-oriented design, such as diagrammatic notations and textual notations. Textual notations can be further divided into formal text, such as logic-based, algebra-based, process-based notations, and informal text (natural languages).

A diagrammatic notation can be beneficial in many ways. First, it can help people grasp large amount of information more quickly than straight text. It can be very compact albeit simple, especially for quantitative information (e.g., a very large set of numbers). A use of diagrams, such as tables, bar charts and pie charts, is usually the simplest and the most powerful method for analyzing and communicating statistical information. Just like an old adage, "A picture is worth a thousand words." Second, graphics can be used for conveying complex concepts and models, such as object-oriented design. Notations like UML are very good at communicating designs given the fact that UML is continuously evolving for better expressiveness as, for example, for frameworks [7, 8]. Third, as well as easy to understand, it is normally easier to learn drawing diagrams than writing text because diagrams are more concrete and intuitive than text written in formal or informal languages.

Unfortunately, there are yet shortcomings of graphical notations. The flip side is that graphical notations are sometimes imprecise, ambiguous, unclear and lack of express power. For example, the UML notations lack the express power for the intuition and essence of design patterns and the hot-spot of frameworks. Although improvements [11] and extensions [7, 8] to UML provide solutions to some particular problems related to the difficulties of modeling non-determinism in UML, losing pattern-related information after the applications and compositions of design patterns remains to be problems of UML. The modeling elements, such as classes, operations, and attributes, in each design pattern usually play some roles that are manifested by their names. The application of a design pattern may change the names of its classes, operations, and attributes to the terms in the application domain. Thus, the role information of the pattern is lost. It is not obvious which modeling elements participate this pattern. As a result, the designer cannot communicate with others about a system design in terms of design patterns used. The benefits of using design patterns compromised. In order to retain the pattern-related information even after the pattern is applied or composed with other patterns, we propose some new notations that extend UML. In our notations, pattern-related information is explicit so that a design pattern can be easily identified when it is applied and composed. The notations are also scalable to large design containing a number of design patterns.

In the following sections, we will first discuss some current techniques for representing the compositions of design patterns. Then, we will describe our approach to the representation of the compositions by extending the UML notations.

2 GRAPHICAL NOTATIONS FOR THE COMPOSITION OF DESIGN PATTERNS

In this section, we present new graphic notations (extensions to UML) to visually represent each individual pattern within an aggregate of patterns. In this work, we do not make the distinction between a composite design pattern[1] and an arbitrary pattern composition. In the following discussions, we use a composition[2] of the Composite pattern [9] and the Decorator pattern [9] to illustrate our notations for representing pattern compositions.

To explicitly represent a pattern in the composition of patterns, we provide notations that are extensions of UML. In this way, each individual pattern is explicit in design documentation so that it can be identified easily. These pattern level notations are as important as (if not more important than) the graphical notations at the class and object level. Before describing the new notations, we discuss some previous methods for explicitly representing individual design pattern in a composition of patterns. We show the pros and cons of these methods and argue that they do not satisfy our expectation.

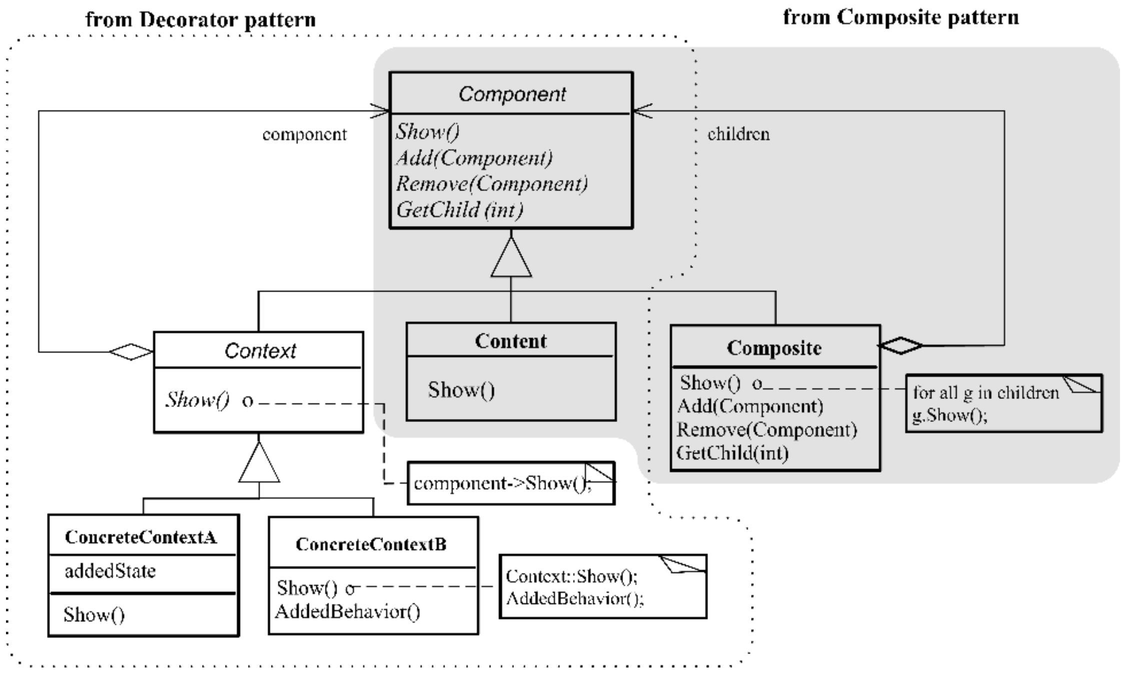

Figure 1: Venn Diagram-Style Pattern Annotation

Venn Diagram-Style Pattern Annotation

The first notation for identifying patterns in a design diagram is based on Venn diagrams [16]. Figure 1 shows the Composite pattern and the Decorator pattern

[1]A composite design pattern is defined as a composition of design patterns in which the resulting composition is also considered to be a design pattern [12, 15]. A composite design pattern can be seen as a special kind of design pattern compositions.

[2]This composition is actually a composite design pattern, called the Navigational Contexts pattern in [13].

manifested themselves in their composition. It shows that the Component, Composite and Content classes participate in the Composite pattern, while Component, Content, Context, ConcreteContextA and ConcreteContextB are participants in the Decorator pattern. This notation works fine with a small number of patterns per class. When a class participates in more and more patterns, the overlapping regions, where the class resides, may become hard to distinguish, especially when different gray levels need to be selected to represent different patterns.

Besides the scalability problem, shading in a diagram has the problem that it may not print well on paper by different printers, nor does it for faxing and scanning. Printers with different quality may render distinctive results when printing dissimilar gray levels.

Another shortcoming of this notation is that it is not explicit what participant roles a modeling element, such as a class, plays. We not only need to identify each pattern in a design diagram, but also want to show the particular roles each modeling element plays.

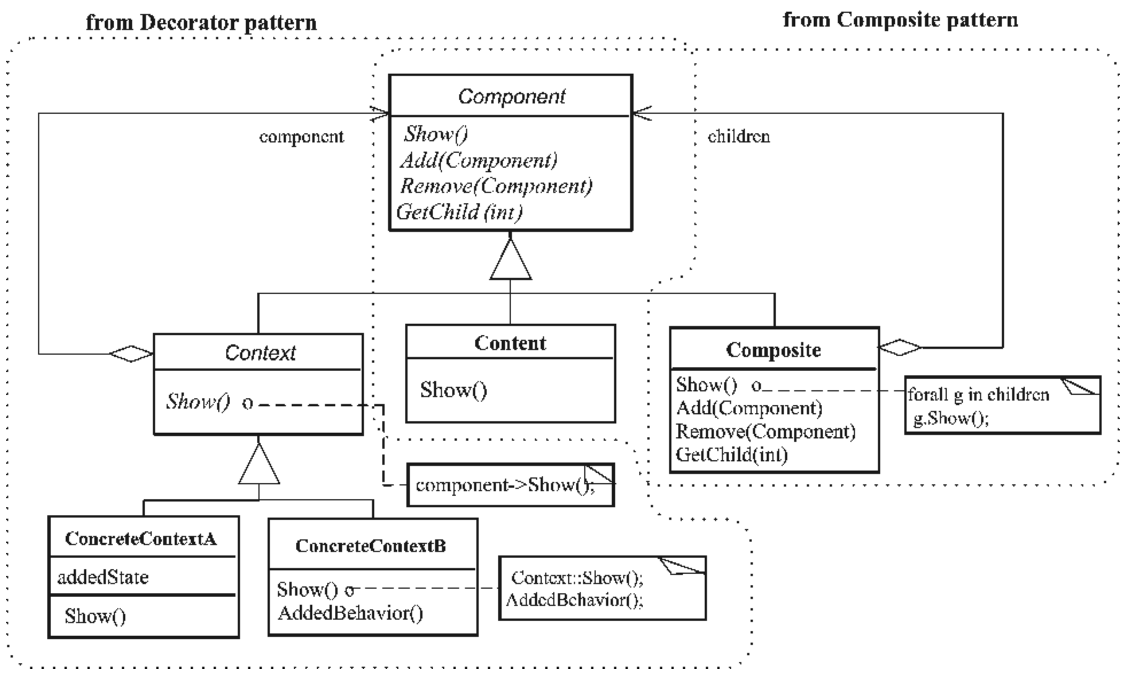

Figure 2: Dotted Bounding Pattern Annotation

Dotted Bounding Pattern Annotation

To prevent the shading problem, we propose a variation of the previous notation that replaces shadings simply by dashed lines. Figure 2 displays the composition of the two patterns, which is similar to Figure 1, except that the shading areas are changed to the regions bounded by dashed lines. This change solves the problem caused by shading. It, yet, remains hard to identify precisely the roles a modeling element, e.g. a class, plays.

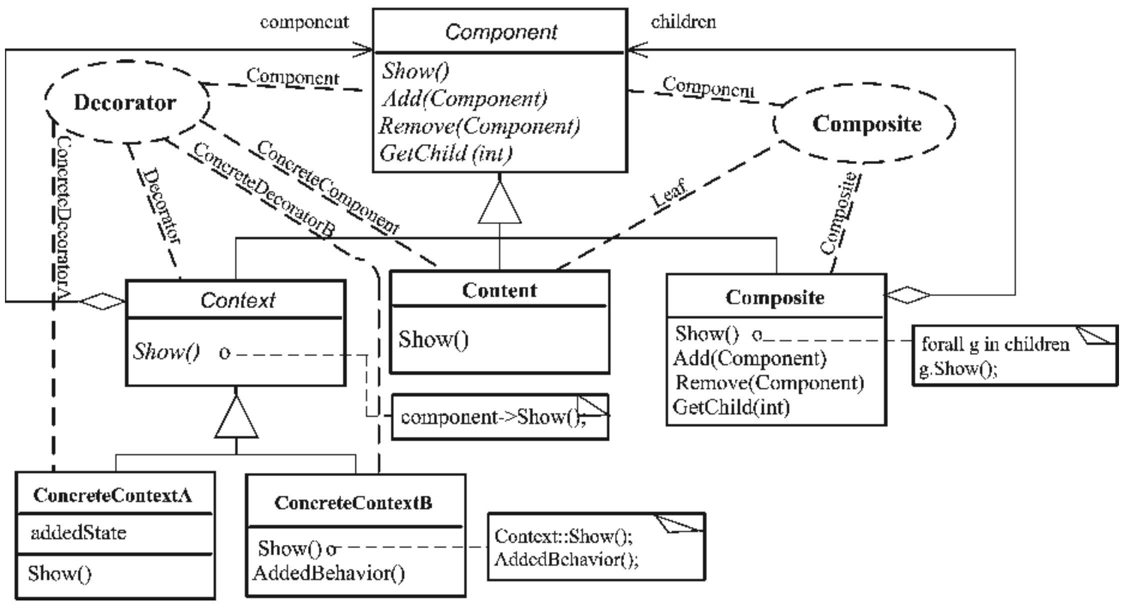

Figure 3: UML Collaboration Notation

UML Collaboration Notation

To address the difficulty of explicit identification of the participant roles a class plays, an alternative notation is provided in UML, called the parameterized collaboration diagrams [14]. This notation can depict design pattern structure by representing patterns and their participants in a class diagram as shown in Figure 3. Dashed ellipses with pattern names inside are used to represent patterns. Dashed lines labeled with participant names are used to associate the patterns with their participating classes. While this notation improves over the previous two notations with the explicit representations of pattern participants, it raises other problems. The dashed lines appear cluttering the presentation. The pattern information is mixed with the class structure, making both hard to distinguish.

Pattern:Role Annotations

To improve the diagrammatic presentation by removing the cluttering dashed lines, Gamma [16] has defined a graphical notation, called “pattern:role annotations”. The idea is to tag each class with a shaded box containing the pattern and/or participant name(s) associated with the given class. If it will not cause any ambiguity, only the participant name is shown for simplification. Figure 4 depicts that the pattern-related annotations appear in shaded boxes as if they are on a different plane from

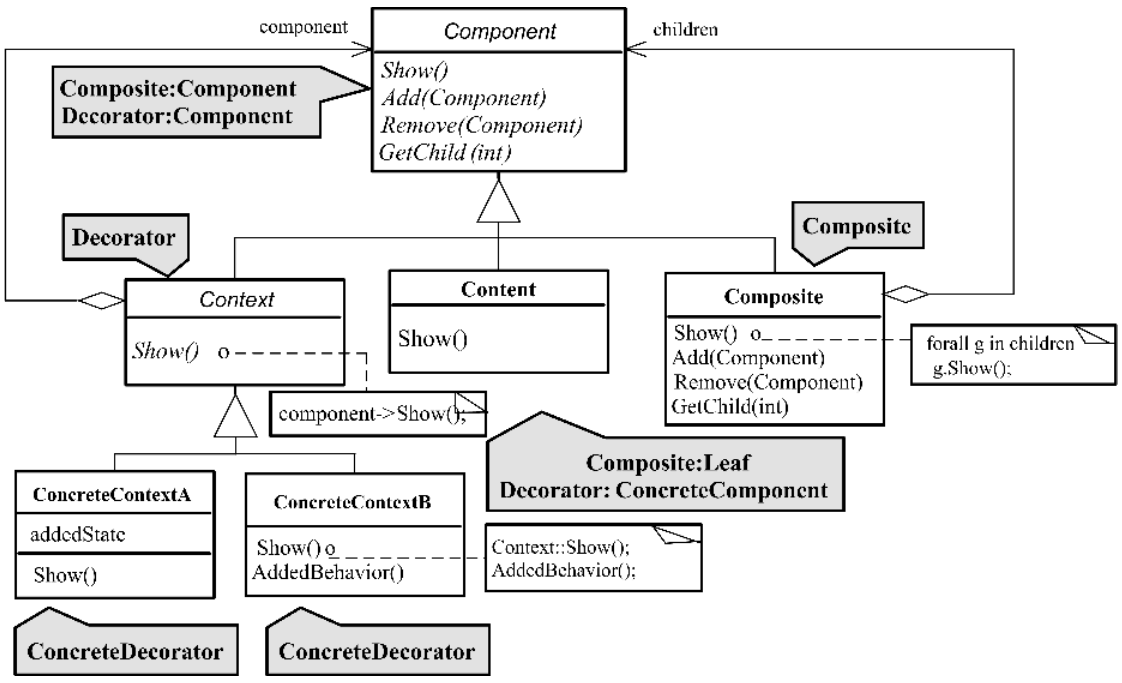

Figure 4: Pattern:Role Annotations

the class structure. This notation is more scalable than the previous notations and highly readable and informative according to [16].

Unfortunately, the problems related shading arise again as the first notation (see Section 2). The gray backgrounds do not fax and scan well. In addition, they may not print well in some printers with low resolution because the gray backgrounds can make the words inside the shaded box illegible. Moreover, not only a class may play some roles in a design pattern, but also an operation (or attribute) may play some roles. This notation fails to represent the roles an operation (attribute) plays in a design pattern.

Stereotype Annotation

Berner et al. [2] proposed a notation based on UML stereotypes, which they called *restrictive stereotype*. A stereotype in UML can add new properties to elements of the underlying language (UML), or can modify existing ones. It is a modeling language, which has a well-formed mechanism for expressing user-definable extensions, refinements or redefinitions of elements of the language without directly modifying the meta-model of the language. Stereotypes provide language users with limited meta-modeling capabilities without giving them direct access to the meta-model of the language. This is very powerful mechanism. The restrictive stereotype is a first-class member in UML, which is defined by rigorous syntax and semantics. The extended stereotype is attached to a class with the pattern name and the role they play in this pattern. Structural restrictions for the pattern language stereotype are defined to constrain on the elements that instantiate the pattern.

According to Berner's classification, all previously presented pattern visualization notations are *descriptive stereotypes* that are on a pure syntactic level. They do not impose any semantic restrictions on the extended or modified syntax. Thus, the advantage of restrictive stereotype is that it is possible to check models for their compliance with both syntax rules and semantic restrictions. With a formal specification, these property checks are automatable [4]. Otherwise, restrictive stereotypes lose much of their power and finally become similar to descriptive ones.

The drawbacks of Berner's stereotype notation include the expensiveness of designing, using and maintaining the notation and the difficulty of scaling it up. Furthermore, the designer of this stereotype notation needs to know, for example, the desired properties of the stereotype to be designed, the base language, the general principles of good language design, and the meta-language that is used to specify the semantics of the stereotype. Ignoring these requirements may result in incomprehensible, contradictory or simply wrong stereotype designs. Incorrectly designed restrictive stereotypes damage the base language rather than improve it. In addition, Berner et al. did not discuss how to extend UML stereotype notation to represent the compositions of design patterns.

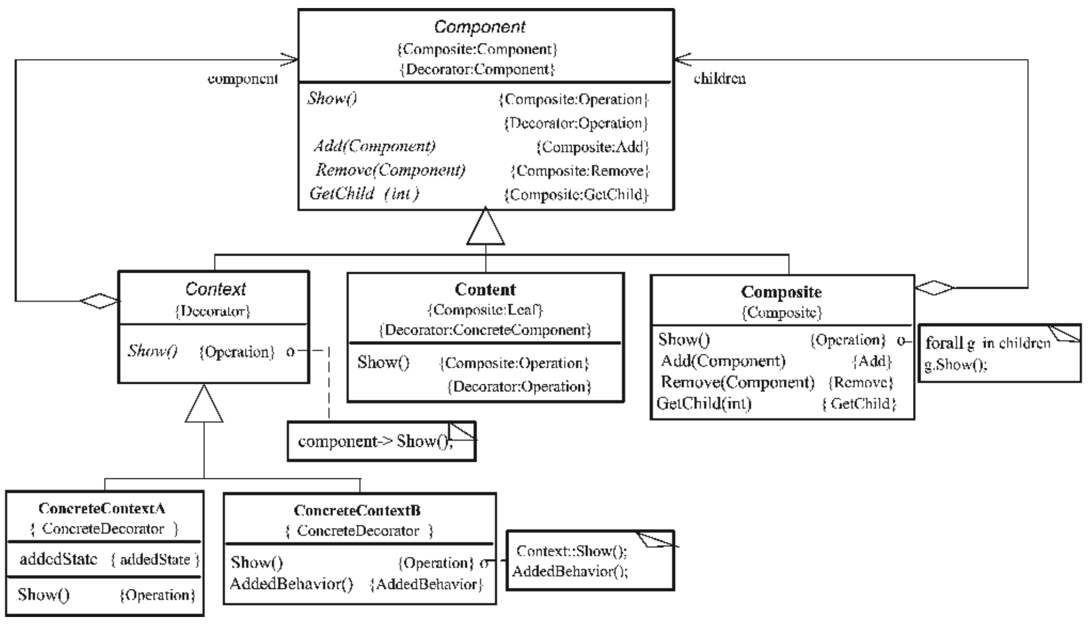

Figure 5: Tagged Pattern Annotation

Tagged Pattern Annotation

Although the Venn diagram-style notation and the dotted bounding notation can show which classes participate a pattern in a design diagram, these notations cannot explicitly represent the role that each class plays in the pattern. The UML collaboration notation and the “pattern:role” notation improve the expressive power by explicitly representing the role that each class plays in the pattern. However, not only a class may play certain role in a pattern, but also an operation (or an attribute) may play certain role in the pattern. None of these notations can represent the information that an operation or an attribute participates and the roles it plays in a pattern. Explicitly representing operation and attribute roles in a pattern is important because many patterns are based on polymorphism, delegation and aggregation, which are often presented based on the relationships among operations and attributes. Explicit representation of the key operations and attributes can not only help on the application (instantiation) of a pattern because the pattern impose some restrictions through the relationships among operations and attributes, but also assist on the traceability of a pattern since it allows us to trace back to the design pattern from a complex design diagram.

In order to represent explicitly the roles of each class, operation, and attribute in a pattern, we propose a new notation that is an extension to UML. The extension is defined mainly by applying the UML built-in extensibility mechanisms. This extension forms a basis for a new UML profile [6], especially useful for representing patterns and their participants. UML provides three language extension mechanisms: stereotypes, tagged values, and constraints. The definition of stereotype can be found in Berner’s notation described previously. Tagged values are used to extend the properties of a modeling element with a certain kind of information. A tagged value is basically a pair consisting of a name (the tag) and the associated value, written as “{tag=value}”. Both tag and value are usually strings only, although the value may have a special interpretation, such as numbers or the Boolean values. In the case of tags with Boolean values, UML allows us to write “{tag}” as a shortcut for “{tag=TRUE}”. A tagged value can be considered as a special kind of stereotype (a descriptive stereotype in the classification in [2]). Model elements can only have one stereotype, but an unlimited number of tagged values that provide more flexibility. We choose to use tagged values because each of the model elements (class, operation and attribute) may participate in different patterns. Constraints may be used to detail how a UML element may be treated. However, like stereotypes and tagged values, constraints have a rather weak semantics and therefore can be used (and misused).

Our new notation is called “tagged pattern annotation”. The idea is that, for each class, we create new tagged values that are used to hold pattern and/or participant name(s) associated with this given class and its operations and attributes. If it will not cause any ambiguity, only the participant name is shown for simplification. Figure 5 displays the diagram based on our notation, where the Component and the Content classes are the overlapping part of the composition of the Decorator pattern and the Composite pattern. With tagged values, the roles that these two classes play in each pattern are shown. In addition, the operations and attributes are attached with tagged values showing the roles they play in each pattern. We found that our notation scales even better than other notations without scarifying readability and informativeness[3]. The limitation of our notation is that the pattern-related information is not as significant as the “pattern:role” notation with shading, which is a trade-off[4]. For a small number of patterns, this new notation can combine with the dotted bounding notation (see Section 2) by bounding each pattern with dashed circles so that the pattern boundaries are explicitly depicted as shown in Figure 7.

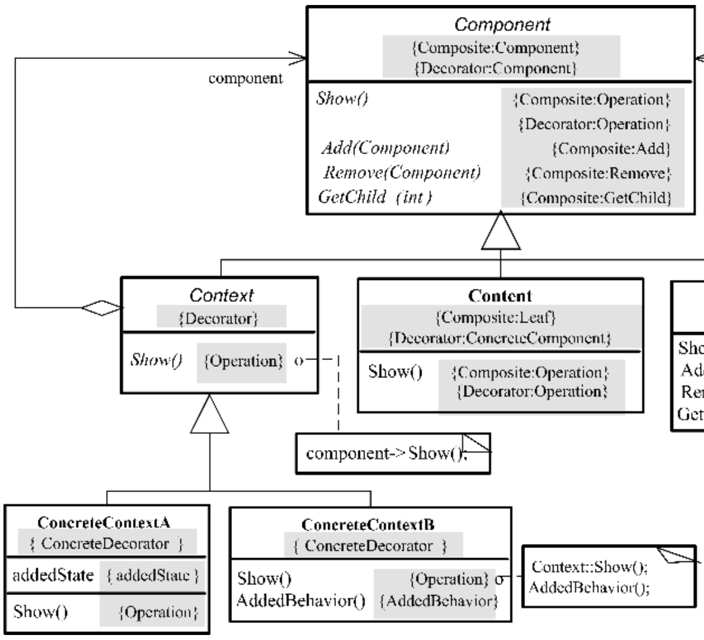

Figure 6: Tagged Pattern Annotation with Shading

Besides the two improvements to make pattern-related information explicit (see Figure 6) and pattern boundaries explicit (see Figure 7), we propose another alternative improvement to extend UML by adding a new compartment in each class in the class diagram. This new compartment of each class is used to hold pattern-related information. Consequently, the pattern and/or participant name(s) associated with a class are put into the new compartment of this given class. In this way, pattern-related information is treated as first-class members in the same way as attributes and operations of a class, as shown in Figure 8, where the roles each class plays are displayed in a separate compartment.

We argue in favor with Berner et al. [2] on that the extension to UML for pattern visualization should not be included in the UML base language because patterns evolve and are frequently application-dependent. Therefore, the set of those patterns that can be used and have to be documented in a model should be

[3]Although the goal of using tagged values is for making pattern-related information explicit and for traceability, many currently available UML case tools give support to reasoning about tagged values and could be adapted to work with our notations.

[4]If we do not need to worry about the shading problem because, for example, everyone has good quality fax machines, scanners, and printers, we can still shade the pattern-related stereotypes so that the pattern-related information appears to occupy a different plane as shown in Figure 6.

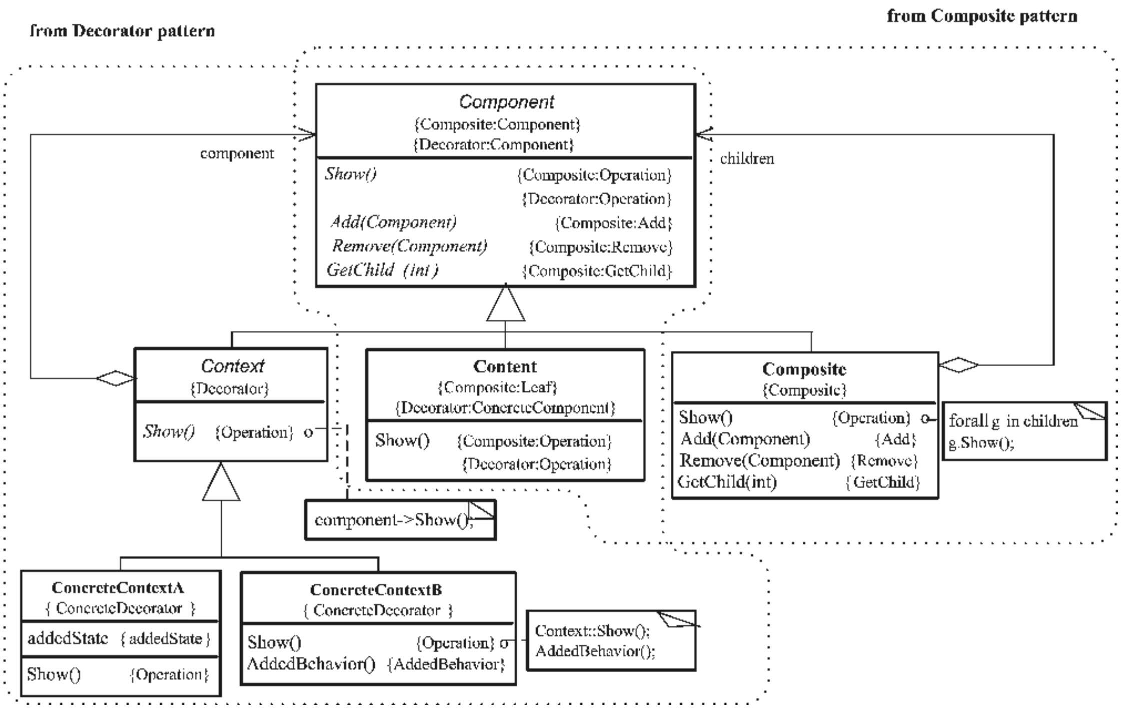

Figure 7: Tagged Pattern Annotation with Bounding

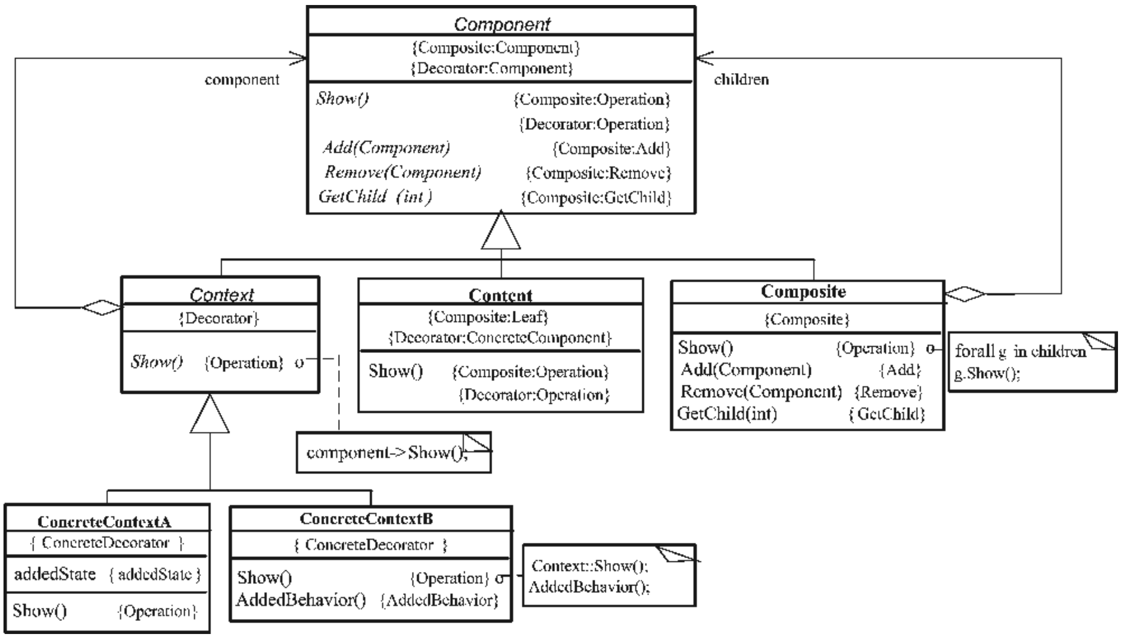

Figure 8: Tagged Pattern Annotation with New Compartments

definable on the level of projects and organizations, instead of part of the UML base language. It should only extend, rather than modify, the syntax and semantics of the base language.

3 CONCLUSIONS

The previously described notations extend the UML class diagram with pattern-related information. The underlying model elements are not changed. They are only attached with some new notations that are not first-class members of the base language. A completely different approach has been proposed by Riehle [12]. This approach is based on role relationships. In this notation, a class, which is not a first-class member, is replaced by a role. A role diagram takes the place of a class diagram in role-based modeling. Role diagrams are better suited for describing object collaboration based patterns than class diagrams, because they focus better on the actual problem solution as a set of collaborating objects. Role diagrams are more abstract than class diagrams, and can be mapped to several class diagrams. A role diagram depicts some collaborating objects that play one or more roles in design patterns. An object collaboration can be viewed as a set of overlapping role diagrams that can be easily composed. Thus, they are attractive for describing composite design patterns. The limitation of this role-based notation is that it is only suitable for describing object collaboration based patterns, making it inappropriate for class inheritance based patterns. In addition, it is not explicitly represented in any object-oriented programming languages. Roles are new concept comparing to classes that have long been the primary means for modeling object-oriented software systems.

In this paper, we introduced some new notations that extend UML to explicitly visualize design patterns. It is important for designers to describe explicitly patterns in a design diagram because the goals of design patterns are to reuse design experience, to improve communication within and across software development teams, to capture explicitly the design decisions made by designers, and to record design tradeoffs and design alternatives in different applications. Currently, the application of a design pattern may change the names of classes, operations, and attributes participating in this pattern to the terms of the application domain. Thus, the roles that the classes, operations, and attributes play in this pattern have lost. This pattern-related information is important to accomplish the goals of design pattern. Without explicitly representing this information, the designers are force to communicate at the class and object level, instead of the pattern level. The design decisions and tradeoffs captured in the pattern are lost too. Therefore, the notations provided in this paper help on the explicit representation of design patterns and accomplishing the goals of design patterns.

REFERENCES

[1] Paulo Alencar, Donald Cowan, Jing Dong, and Carlos Lucena. A Pattern-Based Approach to Structural Design Composition. *Proceedings of the IEEE 23rd Annual International Computer Software & Applications Conference (COMPSAC), Phoenix USA*, pages 160–165, October 1999.

[2] Stefan Berner, Martin Glinz, and Stefan Joos. A Classification of Stereotypes for Object-Oriented Modeling Languages. *Proceedings of the Second International Conference on the Unified Modeling Language (UML), LNCS1723, Springer-Verlag*, pages 249–264, October 1999.

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison Wesley, 1999.

[4] Jing Dong. Design Component Contracts: Model and Analysis of Pattern-Based Composition. *Ph.D. Thesis, Computer Science Department, University of Waterloo*, June 2002.

[5] Jing Dong, Paulo Alencar, and Donald Cowan. Ensuring Structure and Behavior Correctness in Design Composition. *Proceedings of the 7th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems(ECBS), Edinburgh UK*, pages 279–287, April 2000.

[6] Desmond D'Souza, Aamod Sane, and Alan Birchenough. First Class Extensibility for UML – Packaging of Profiles, Stereotypes, Patterns. *Proceedings of the Second International Conference on the Unified Modeling Language (UML), LNCS1723, Springer-Verlag*, pages 265–277, October 1999.

[7] Marcus Fontoura and Carlos Lucena. Extending UML to Improve the Representation of Design Patterns. *Journal of Object Oriented Programming*, 13(11):12–19, March 2001.

[8] Marcus Fontoura, Wolfgan Pree, and Bernhard Rumpe. UML-F: A Modeling Language for Object-Oriented Frameworks. *Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP)*, pages 63–82, July 2000.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison Wesley Publishing Company, 1995.

[10] Rudolf K. Keller and Reinhard Schauer. Design Components: Towards Software Composition at the Design Level. *Proceedings of the 20th International Conference on Software Engineering*, pages 302–311, 1998.

[11] Anthony Lauder and Stuart Kent. Precise Visual Specification of Design Patterns. *Proceedings of the 12th European Conference on Object-Oriented Programming (ECOOP)*, pages 114–134, July 1998.

[12] Dirk Riehle. Composite Design Patterns. *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA), USA*, pages 218–228, October 1997.

[13] Gustavo Rossi, Daniel Schwabe, and Alejandra Garrido. Design Reuse in Hypermedia Applications Development. *Proceedings of the ACM International Conference on Hypertext*, pages 57–66, April 1997.

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

[15] John Vlissides. Composite Design Pattern (They Aren't What You Think). *C++ Report*, June 1998.

[16] John Vlissides. Notation, Notation, Notation. *C++ Report*, April 1998.

ABOUT THE AUTHOR

Jing Dong is an assistant professor in the Computer Science Department at the University of Texas at Dallas. He received a Ph.D. in Computer Science from the University of Waterloo. His research interests include design patterns, UML, component-based software engineering, and formal methods. He can be reached at jdong@utdallas.edu and http://www.utdallas.edu/∼jdong.