

VisDP: A Web Service for Visualizing Design Patterns on Demand

Jing Dong, Sheng Yang and Kang Zhang
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083, USA
{jdong,syang,kzhang}@utdallas.edu

Abstract

Design patterns document good design solutions to a recurring problem in a particular context. They are typically modeled using UML. In practice, however, pattern-related information is lost when a design pattern is applied or composed because UML does not keep track of this information. Consequently, the designer cannot identify design patterns used in software system design diagrams. The benefits of design patterns are compromised because the designers cannot communicate with each other in terms of the design patterns they use and their design decisions and tradeoffs. In this paper, we present a web service (VisDP) for explicitly visualizing design patterns in UML diagrams. This web service is developed based on a UML profile containing new stereotypes, tagged values and constraints for visualizing design patterns in UML diagrams. With this service, the user is able to identify design patterns by moving the mouse and viewing color changes in UML diagrams. Additional pattern-related information can be dynamically displayed based on the mouse location.

Keywords: Design pattern, UML, web service, visual tool, software visualization.

1. Introduction

Design patterns [5] have become common building blocks to design large-scale software systems. A pattern is a recurring solution to a standard problem. It has a context in which it applies. Design patterns are usually modeled and documented in the Unified Modeling Language (UML) [1]. However, the constructs provided by the standard UML are unable to visualize design patterns in their applications and compositions. In the UML diagrams of a design pattern, the model elements, such as classes, operations, and attributes, usually play certain roles that are manifested by their names. These names may be changed to the terms in the application domain when the design pattern is applied or composed with other patterns. Thus, the role information of the pattern is lost. It is not obvious which model elements participate in this pattern so that it is hard for a designer to identify design patterns in software system designs [11].

There are several problems when design patterns are implicit in software system designs: first, software developers can only communicate at the class/object level instead of the pattern level since they do not have pattern-related information in system designs. The benefits of design patterns are compromised because the designers cannot communicate with each other in terms of the design patterns they use and their design decisions and tradeoffs. Second, each pattern often documents some ways for future evolutions, which are buried in system designs. Third, it may require considerable efforts on reverse-engineering design patterns from software system designs [6].

Several approaches have been proposed to solve this problem [11][1][4]. However, all of these solutions tend to attach static notations and/or information on UML diagrams, which may inflate the original diagram with pattern-related information. In this paper, we provide visual techniques and a tool, called VisDP, which is able to visualize pattern-related information in a UML diagram on demand. With VisDP, the user is able to identify design patterns by moving the mouse and viewing color changes in UML diagrams. Additional pattern-related information can be dynamically displayed based on the current mouse location. VisDP is developed based on a UML profile containing new stereotypes, tagged values and constraints for visualizing design patterns in UML diagrams.

Service-Oriented Architecture (SOA) [17] is an architectural style whose objective is to reduce coupling among interacting software agents. A web service [2] is a SOA with additional constraints: first, interfaces must be based on Internet protocols such as HTTP, FTP, and SMTP. Second, messages must be in XML except for binary data attachment. VisDP is deployed as a web service so that anyone can use it on-line by accessing our tool website [16] and providing, as an input, a XML file generated by the plug-ins of a UML tool, such as Rational Rose [15] or ArgoUML [14]. In this way, we present a service-oriented architecture that not only allows existing services to work with current UML tools but also allows new services to be developed. With the continuing globalization of software industry, software project development becomes a global task involving the collaborations of developers from different locations. VisDP can be used to support cooperative group work.

The remainder of this paper is organized as follows. In the next section, we present our proposal for a UML profile. In Section 3, we describe our techniques for on-demand visualization. In Section 4, we discuss our web service. In the last two sections, we discuss related work and conclude this paper.

2. A UML Profile

A UML profile is a stereotyped package containing model elements that have been customized for a specific domain or purposed by extending the meta-model using stereotypes, tagged values and constraints. A stereotype, denoted by <<stereotype-name>>, allows the definition of extensions to the UML vocabulary. It groups tagged values and constraints under a meaningful name. When a stereotype is branded to a model element, the semantic meaning of the tagged values and the constraints associated with the stereotype are attached to that model element implicitly. Tagged values extend model elements with new kinds of properties with the format of a pair of name and an associated value, i.e., {name=value}. Constraints add new semantic restrictions to a model element by the Object Constraint Language (OCL) [12].

In the following, we introduce our UML extensions through an example, summarize the new stereotypes, tagged values and constraints, and present a general description of their semantics. We also present a description of how the UML extensibility mechanisms have been applied in the definition of a UML profile for design patterns.

Figure 1 shows a system design that manages the connections to different types of databases, such as Oracle and MySQL (www.mysql.com). This system provides a connection pool for accessing each type of database. The connection pool restricts a limit number of accesses to a database and reuses connections to the database. The system has the capability to handle different types of database connections. The ConnectionPool class defines an interface for the creation of a connection pool for the appropriate type of database. The concrete classes, OracleConnectionPool and MySQLConnectionPool, use the createConnection operation to create the corresponding connections, OracleConnection and MySQLConnection, respectively. All connection instances have the same interface which is defined in the Connection class.

There are two design patterns, Abstract Factory and Singleton, applied in the system design¹. The ConnectionPool, OracleConnectionPool and MySQLConnectionPool classes play the roles of abstract and concrete factories,

whereas the Connection, OracleConnection and MySQLConnection classes play the roles of abstract and concrete products in the Abstract Factory pattern, respectively. OracleConnectionPool and MySQLConnectionPool are the Singleton classes, which restrict only one connection pool for each database.

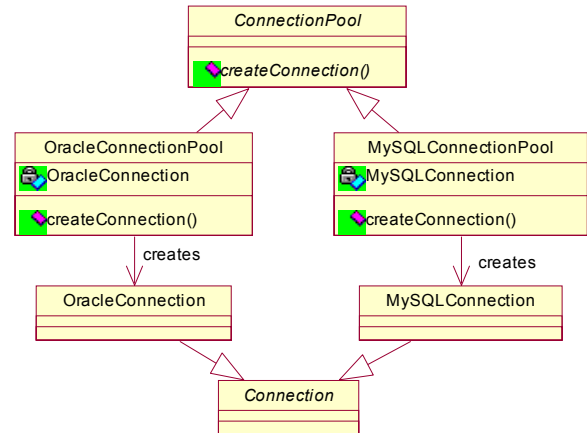


Figure 1 Connection Pool

To explicitly visualize design pattern structure and behavior in class and collaboration diagrams, respectively, we define a UML profile which includes three stereotypes: PatternClass, PatternAttribute and PatternOperation, whose base classes are Class, Attribute, and Operation, respectively. Each stereotype also defines one tagged value which defines exactly what role a class, an attribute or an operation plays in a design pattern. The name of the tagged value is “pattern” and the value of the tagged value is a tuple in the format of <name:string[instance:integer], role:string>. The “name” in the tuple is the pattern name in which a model element, such as class, attribute or operation, participates. The name field of PatternAttribute and PatternOperation can be omitted if the class plays a role only in one pattern, and this omission will not create any ambiguity. But the name field of PatternClass is mandatory. Sometimes there is more than one instance of a pattern in the system, and it is useful to distinguish the instances. The “instance” in the tuple indicates the instance of the pattern the model element participates. The “role” in the tuple shows the role that a model element plays in the pattern.

A model element may simultaneously play different roles in different patterns. In this case, a new tagged value with the format as <name[instance],role> is branded to the model element for each additional pattern it participates.

The constraint of the PatternClass stereotype is defined formally in OCL as follows:

```

<<PatternClass>>:
self.baseClass = Class and self.taggedValue -> exists
(tv:taggedValue | tv.name = "pattern" and tv.dataValue =
"tuple<name:string[instance:integer],role:string>")
  
```

¹ To illustrate our approach, we use this small example. It may be not hard, if not obvious, to discover the two design patterns used in this example. It is not easy to identify design patterns in a larger system design with many patterns.

where the base class of the PatternClass stereotypes is Class. It has a tagged value with the name of “pattern” and the value of “name[instance],role”. The types of “name” and “role” are string and the type of “instance” is integer. The constraints of the PatternAttribute, and PatternOperation stereotypes are defined similarly. These stereotypes, together with their tagged values and constraints, form a new UML profile for design patterns.

Figure 2 is a UML diagram attached with new stereotypes and tagged values to explicitly visualize one instance of the Abstract Factory pattern and two instances of the Singleton pattern. Consider, for example, the OracleConnectionPool class which has an attribute OracleConnection and an operation createConnection() shown in Figure 2. This class participates in both the Abstract Factory and the

Singleton patterns. It plays the role of “ConcreteFactory” in the Abstract Factory pattern and the role of “Singleton” in the first instance of the Singleton pattern. Thus, the stereotype <<PatternClass { <Abstract Factory, ConcreteFactory> <Singleton[1], Singleton> } >> is attached to the OracleConnectionPool class. Similarly, since the OracleConnection attribute only participates in the first instance of the Singleton pattern, it is only attached by the <<PatternAttribute{<Singleton[1], UniqueInstance}>> stereotype. Since the createConnection() operation, plays the role of “createProduct” in the Abstract Factory pattern and the role of “instance” in the first instance of the Singleton pattern, a <<PatternOperation {<Abstract Factory, createProduct> <Singleton[1], Instance>} >> stereotype is attached to it.

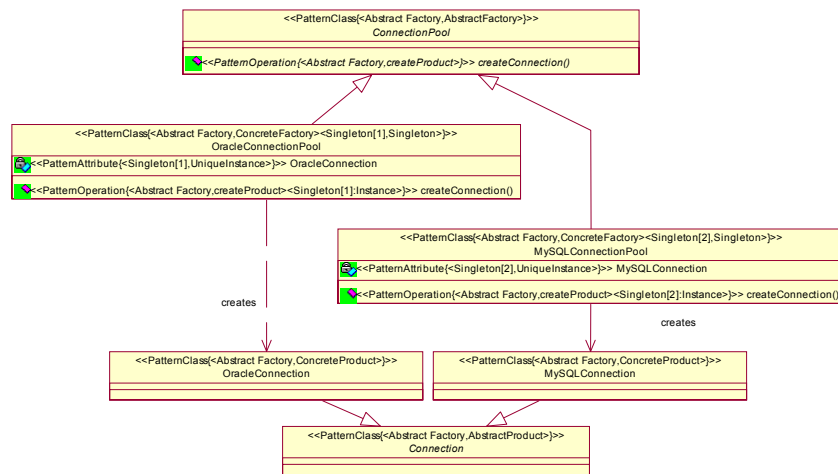


Figure 2 Connection Pool Modeled with the New Stereotypes and Tagged Values

3. On-Demand Visualization

As discussed in the previous section, we have extended UML with a profile. Stereotypes and tagged values are attached to UML diagrams to display pattern-related information. Although our UML profile provides more comprehensive pattern-related information and is easier to incorporate with a current UML tool, the additional pattern-related information does not scale well with large applications as shown in Figure 2. The main problem with current solutions to visualizing design patterns is that static pattern-related notations or information is attached to traditional UML diagrams, leading to the following consequences. First, the pattern-related information is tangled with the class structure, making them both harder to see. Second, the additional static information does not scale well in large UML diagrams. Third, the user is not able to concentrate on a particular part of a diagram for visualization.

To solve these problems, we propose on-demand visualization techniques based on coloring and mouse movement. We also develop a tool, called VisDP, which can hide/show pattern-related information on demand.

When the pattern-related information is hidden, the diagrams are just like the ordinary UML diagrams. When the pattern-related information is shown, the end user can identify the design patterns that a class (operation/attribute) participates and the roles it plays by different colors and on-demand information shown in the diagram. By using our techniques and tool in a UML diagram, the software designer can move his/her mouse onto the modeling element (e.g., class, operation, attribute) in question. All classes that participate in the same pattern as the class under the mouse are changed to the same color. If the class under the mouse participates in more than one design pattern, different colors are used to distinguish the patterns. In the overlapping part of a composition of patterns, the corresponding colors of all participating patterns are displayed in alternation with a certain time interval. Outside the overlapping part, only the cor-

responding colors are displayed. All pattern-related information encoded in the stereotypes and tagged values of a modeling element is shown when the user moves the mouse over that modeling element. When the user's mouse is moved out, all pattern-related information is gone and no color is shown. The diagram returns back to normal.

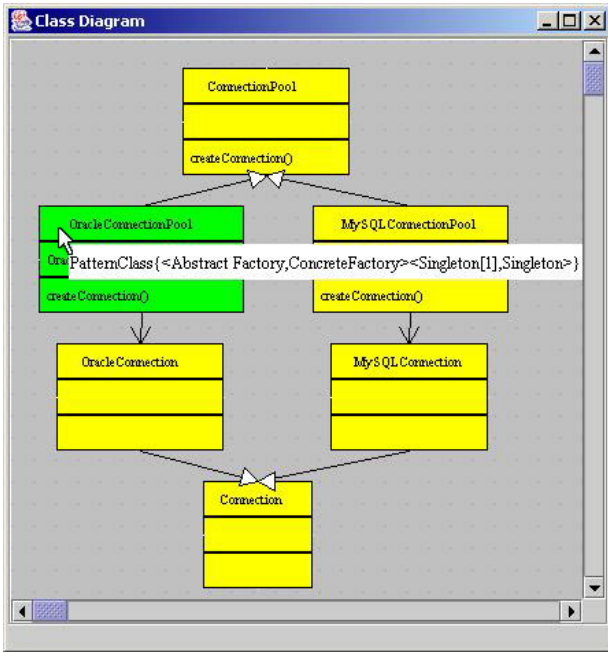


Figure 3 Connection Pool Class Diagram with Pattern Information Shown

Figure 3, for instance, demonstrates the scenario when a software designer moves his/her mouse over the OracleConnectionPool class in the example UML diagram. Since this class participates in both the Abstract Factory pattern and the Singleton pattern, two colors are displayed. All classes participate in the Abstract Factory pattern are changed to one color (yellow, appeared as light-grey in black/white printout), whereas the class participates in the Singleton pattern is changed to the other color (green, appeared as dark-grey in black/white printout). The color of the overlapping class(es) (OracleConnectionPool) is alternated between yellow and green with a certain time interval. Since the designer's mouse is over the class name compartment of the OracleConnectionPool class, VisDP shows a text box containing detailed pattern-related information described in terms of the stereotype and tagged values of this class as discussed in the previous section. Through the stereotype and tagged values, therefore, the designer is able to identify this class participates in both the Abstract Factory and the Singleton patterns. It plays the role of "ConcreteFactory" in the Abstract Factory pattern and the role of "Singleton" in the first instance of the Singleton pattern. Suppose the designer moves the mouse down to the operation compart-

ment of the same class, the stereotype and tagged values of the operation createConnection() are displayed. All colors are unchanged. When the user's mouse is moved out, all pattern-related information and colors disappear. The diagram returns back to normal as shown in Figure 2.

4. A Web Service for Visualizing Patterns

In the previous section, we describe our techniques and a tool for visualizing design patterns in terms of standalone applications. In this section, we present how to provide a web service for our techniques to be broadly accessible in the Internet and how to incorporate our tool with common UML tools, such as Rational Rose and ArgoUML. We use service oriented architecture to publish our application which is based on the following considerations. First, the web service is accessible through a web browser so that the user can access our application anywhere. Second, web service provides the communication between applications. VisDP can not only be accessed by web browsers, it can also be plugged into other standalone applications.

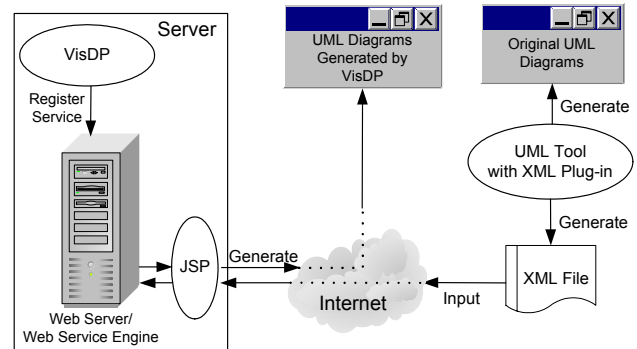


Figure 4 Overall System Architecture

Figure 4 depicts the overall system architecture of VisDP. On the server side, we deploy VisDP as a web service which is registered in the web service engine so that it is ready to serve users. On the client side, the user may use any common UML tools, such as Rational Rose, to draw diagrams with stereotypes and tagged values representing pattern-related information as our discussions in Section 2. In this case, the stereotypes and tagged values are hidden. The diagrams are normal UML diagrams. When the user wants to use VisDP web service to visualize pattern-related information encapsulated in the diagrams, he/she may use an appropriate plug-in (e.g. UniSys XMI for Rational Rose) to transform a UML diagram into a XML file. The JSP page of VisDP takes the XML file as an input and returns the user with a new UML diagram that can visualize design pattern on demand as shown in Section 3.

There are several advantages of this system architecture of VisDP. First, it works with current UML tools, not only Rational Rose, as long as a corresponding plug-in can generate the XML files of the UML diagrams. Second, new services can be provided as, for example, new services for visualizing UML diagrams other than class and collaboration diagrams. Third, our service can be used anywhere and anytime.

In short, the overall process can be summarized as the following process: registering web service, generating XML file, invoking the service and generating new UML diagram.

In order to be accessible from remote clients, the VisDP web service has to be registered to a web service engine by providing the description of the service. The deployment descriptor is illustrated as following:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment" id="urn:UmlClassDiag">
  <isd:provider type="java"
    scope="Application"
    methods="drawUMLClassDiagram">
    <isd:java class="cs.MainService" static="false"/>
  </isd:provider>
</isd:provider>
<isd:faultListener>
  org.apache.soap.server.DOMFaultListener
</isd:faultListener>
</isd:service>
```

The “id” attribute in <isd:service> tag refers to the service name, which is known to the client. Tag <isd:service> contains two parts: provider and faultListener. Tag <isd:provider> provides the detailed information of a service. The “type” attribute refers to the type of the service. The “scope” attribute specifies the lifetime of the service object. We set “scope” to “application” which means that we want the service object to be instantiated once and everyone can share this object from then on. The “methods” attribute specifies the method name which is available to the client. Subtag <isd:java> is used to set the service type to Java. Attribute “class” in <isd:java> simply specifies the name of a Java class file to respond the client’s requests. Tag <isd:faultListener> is a sibling of <isd:provider>, which deals with the exceptional situation when the client’s requests cannot be fulfilled for some reason.

The web service interface is described in Web Service Description Language (WSDL) [18]. The following items are defined in this interface:

- **Message.** There is one input message that is defined for the service. The type of the message is string. The value of this string represents the file name of the XML file corresponding to the UML class diagram. This XML file is generated by the plug-ins of UML tools.

```
<wsdl:message name='drawUMLClassDiagramIn'>
```

```
<wsdl:part name='filename' type='xsd:string'/>
</wsdl:message>
```

- **Operation.** There is one operation (drawUMLClassDiagram) exposed to the public users. It is defined in the portType tag in the WSDL file. The operation takes a string as its input.

```
<wsdl:portType name='MainService'>
  <wsdl:operation name='drawUMLClassDiagram'
    parameterOrder='filename'>
    <wsdl:input name='drawUMLClassDiagramIn' message='tns:drawUMLClassDiagramIn'/>
  </wsdl:operation>
</wsdl:portType>
```

- **Service.** The name of the service is MainService, its address is ‘http://129.110.96.185:8080/soap/servlet/rpcrouter’.

```
<wsdl:service name='MainService'>
  <wsdl:port name='MainService' binding='tns:MainService'>
    <soap:address location='http://129.110.96.185:8080/soap/servlet/rpcrouter'/>
  </wsdl:port>
</wsdl:service>
```

A UML diagram is transformed into XML format to be passed to the VisDP web service. This transformation can be achieved by plug-ins to UML tools, such as Unisys XMI plug-in for Rational Rose. Figure 5 illustrates part of the XML file generated by the plug-in from the UML diagram shown in Figure 2, where the ConnectionPool class has an operation “createConnection”. Both the class name and the operation name are highlighted in Figure 5. In addition, the stereotype names of PatternClass and PatternOperation are highlighted. A stereotype PatternClass{<Abstract Factory, AbstractFactory>}, whose base class is “Class”, is branded to the ConnectionPool class. A stereotype PatternOperation{<Abstract Factory, createProduct>}, whose base class is “Operation”, is attached to operation “createConnection”. These XML files are passed as messages to the VisDP which parses all pattern-related information recorded in the XML files for dynamically visualizing design patterns in a UML diagram.

The user can invoke the service through the JSP page provided by VisDP. In the JSP page, the user provides the XML files generated by a UML plug-in as discussed previously. The JSP page passes the XML message to the web service which receives the message and begins to process the message. A Java applet, which allows the user to dynamically visualize pattern-related information, is returned to the user. The user can visualize pattern-related information by moving the mouse over the UML elements as described in Section 3.

VisDP takes a XML file from the JSP page and generates a new UML diagram that can visualize design pat-

terns on demand. The XML file, for example, shown in Figure 5, contains the information of all modeling elements of a class diagram, such as classes, attributes, and operations, all relationships between entities, such as dependencies, generalizations, and associations, and stereotypes attached to modeling elements. The XML file also has information about the geometric position of each entity. In addition, the XML file contains the information of stereotypes and tagged values defined in the original UML diagrams, which document the pattern-related information. VisDP parses this information and generates a new UML diagram that is the same as the original diagram enhanced with the on-demand visualization capability. In Section 3, we present an example of such UML diagrams.

```

<UML.Namespace.ownedElement>
<!-- ===== connection_pool_tag::ConnectionPool [Class] ===== -->
<UML.Class xmi.id = 'S.213.1352.07.1'
name = 'ConnectionPool' visibility = 'public' isSpecification = 'false'
isRoot = 'true' isLeaf = 'false' isAbstract = 'true'
isActive = 'false'
namespace = 'G.0'
specialization = 'G.2.G.3' >
<UML.Classifier.feature>
<!-- = connection_pool_tag::createConnection [Operation] ===== -->
<UML.Operation xmi.id = 'S.213.1352.07.2'
name = 'createConnection' visibility = 'public' isSpecification = 'false'
ownerScope = 'instance'
isQuery = 'false'
concurrency = 'sequential' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' specification = '' >
<UML.Classifier.feature>
<UML.Class>
<UML.Stereotype xmi.id = 'S.213.1352.09.0'
name = 'PatternClass<Abstract Factory, Abstract Factory>' visibility = 'public' isSpecification = 'false'
isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
icon = '' baseClass = 'Class'
extendedElement = 'S.213.1352.07.1' >
<UML.Stereotype xmi.id = 'S.213.1352.09.1'
name = 'PatternOperation<Abstract Factory, createProduct>' visibility = 'public' isSpecification =
'false'
isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
icon = '' baseClass = 'Operation'
extendedElement = 'S.213.1352.07.2' >
<!-- ===== connection_pool_tag::OracleConnectionPool [Class] ===== -->
<UML.Class xmi.id = 'S.213.1352.07.3'
name = 'OracleConnectionPool' visibility = 'public' isSpecification = 'false'
isRoot = 'false' isLeaf = 'true' isAbstract = 'false'
isActive = 'false'
namespace = 'G.0' clientDependency = 'G.6'
generalization = 'G.2' >

```

Figure 5 Input XML File for Class Diagram

5. Related Work

Explicitly visualizing design patterns in UML has been investigated in [11], where all approaches can only represent the role a class plays in a pattern and cannot represent the role an attribute (or operation) plays in a pattern. They cannot distinguish multi-instance of a pattern either. These approaches provided new notations and/or text information attached to the UML diagrams to represent pattern-related information. Nevertheless, the additional information is static which does not scale well. In this paper, we present a UML profile that can represent the role an attribute (operation), as well as a class, plays in a design pattern and distinguish multi-instance of a design pattern. In addition, we hide all pattern-related information and allow the user to visualize design pattern

on demand. In this case, pattern-related information is displayed only when requested.

France *et al.* [4] specialized the UML metamodel to obtain a pattern specification. Pattern-related information is defined as roles in subtypes of UML metamodel. The application of a design pattern is mapped to its specification by dotted line with arrow head. In this case, pattern specifications always need to be presented to be able to visualize pattern-related information in a UML diagram. The pattern specification diagram and the dotted lines bound to the corresponding pattern application are static information added on normal UML diagram, which prevents the approach from scaling up. There is no discussion on the composition of patterns and multi-instance of patterns either.

Lander and Kent [7] specified a design pattern in type-model and role-model in addition to class-model to tackle the impure pattern modeling problem that is the difficulty of expressing non-deterministic number of concrete classes using UML. Although their notations are quite expressive, they suffer scalability. When several patterns are composed, the three models and the mapping among them become very complex. Furthermore, it requires considerable learning curve due to the complexity of their notations.

Tool support for applying (forward engineering) and discovering (reverse engineering) design patterns has been developed in [3]. This tool is based on the fragment model and fragment database. Although the fragment structure diagram can be used to visualize the role each class (attribute or operation) plays in a design pattern, it does not keep the topology of the original UML diagram so that class model information is lost. Their tool cannot visualize a program purely in terms of pattern instances.

Reiss [9] proposed a specification language for defining design patterns, which breaks a design pattern down into elements and constraints over a program database of structural and semantic information. Each system has a database to store the design patterns defined in this language. Design pattern instances can be created, found, maintained, and edited by querying the database. Based on this language, a tool is developed to allow the user to identify and create pattern instances in the source code. The objective of this approach is to facilitate the application and discovery of design patterns, instead of visualizing them.

Meta-level collaborations and constraints have been used to precisely represent structural and behavioral constraints of design patterns in [8][10]. Structural constraints are expressed by attaching stereotypes <<Clan>>, <<Tribe>> to model elements, whereas behavioral constraints are expressed by temporal logic. A tool which can automatically implement design patterns in a system was also developed.

Unlike these previous works, our goal is to explicitly visualize the hidden dependency of the design pattern to its instance link. We also support design pattern composition, dynamic aspects of design patterns, OCL-style constraint language, the overlapping of design pattern instances, and dynamic pattern visualization.

6. Conclusion

In this paper, we introduce techniques and a tool for explicit visualization of design patterns in system designs. 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. Without explicitly representing pattern-related information, the designers are forced to communicate at the class and object level, instead of the pattern level. The design decisions and tradeoffs captured in the pattern are also lost.

There are several benefits in our approach. First, the readability is improved. The new UML diagram generated by VisDP separates the class and object structures from the pattern-related information. Thus, the pattern-related information no longer distracts the user's attention in the whole system design. Second, our dynamic visual approach maintains scalability in that pattern-related information is implicit. Thus, all UML diagrams with pattern-related information take the same space as those without pattern-related information. There is no additional space required to accommodate this additional information. The diagrams remain the same size on display. Third, the new UML diagram generated by VisDP is adaptable to user's desire. The user can choose to either show or hide pattern-related information according to his/her focus. If the focus is on the overall system design and the relationship between the UML elements (such as classes, operations, and attributes), the pattern-related information can be hidden to avoid distracting the user's attention. On the other hand, if the user wants to know the design patterns used in the system, the participants of each pattern, and the roles of each UML elements, the pattern-related information can be shown to help the user to identify these kinds of information. Fourth, VisDP works complementarily with current UML tools, such as Rational Rose. It does not replace these tools. Since the main goal of VisDP is to dynamically visualize design patterns, it does not provide the capabilities, such as code generation and consistency checking. The user can easily switch back to current UML tools for these purposes. Software visualization is usually not an independent task. It complements other software development tasks, such as modeling, design, and analysis. To this end, we incorpo-

rate design pattern visualization with current design tools so that our techniques and tools are not standalone. Fifth, deploying VisDP as a web service allows the user to access it anywhere and anytime. Clearly understanding design patterns is important for both system architects and developers. It is helpful for them to better understand the application of the design patterns in system designs and the composition of patterns. Making the application accessible to all users allows them to take advantage of the separation of the pattern-related information and UML class diagram.

References

- [1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- [2] E. Cerami. *Web Services Essentials*. O'Reilly & Associates, Inc, 2002
- [3] G. Florijn, M. Meijers, and P. van Winsen. "Tool Support for Object-Oriented Patterns" *Proceedings of European Conference on Object-Oriented Programming*, 1997.
- [4] R. B. France, D. Kim, S. Ghosh, and E. Song. "A UML-Based Pattern Specification Technique" *IEEE Transactions on Software Engineering*, Vol. 30, No. 3, March 2004.
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, 1995.
- [6] R. Keller, R. Schauer, S. Robitalille, and P. Page. "Pattern-Based Reverse-Engineering of Design Components" *Proceedings of the 21st International Conference on Software Engineering*, pages 226-235, May 1999.
- [7] A. Lauder and S. Kent. "Precise Visual Specification of Design Patterns" *Proceedings of European Conference on Object-Oriented Programming*, 1998, p114-134.
- [8] A. LeGuennec, G. Sunye, and J. Jezequel. "Precise Modeling of Design Patterns" *Proceeding of International conference on the Unified Modeling Language*, October 2000.
- [9] S. P. Reiss. "Working With Patterns and Codes" *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000.
- [10] G. Sunye, A. LeGuennec, and J. Jezequel. "Design Patterns Application in UML" *Proceedings of European Conference on Object-Oriented Programming*, 2000, p 44-62
- [11] J. Vlissides. *Notation, Notation, Notation. C++ Report*, April 1998.
- [12] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.
- [13] S. M. Yacoub and H. H. Ammar. "UML Support for Designing Software Systems as a Composition of Design Patterns" *Proceedings of the International Conference on Unified Modeling Language*, Springer-Verlag, 2001.
- [14] ArgoUML website. <http://argouml.tigris.org/>
- [15] Rational Rose website. <http://www.rational.com/>
- [16] VisDP. <http://www.utdallas.edu/~jdong/VisDP>
- [17] Web Services Architecture Requirements, W3C Working Draft 14, November 2002. <http://www.w3.org>
- [18] Web Services Description Language (WSDL), 1.1, W3C Note 15 March 2001. <http://www.w3.org/TR/wsdl>