Enhanced Design Pattern Definition Language

Conference Paper · October 2014

CITATION
1

READS
608

2 authors:

Salman A. Khwaja
King Fahd University of Petroleum and Minerals
7 PUBLICATIONS **33** CITATIONS
SEE PROFILE

Mohammad Alshayeb
King Fahd University of Petroleum and Minerals
96 PUBLICATIONS **1,834** CITATIONS
SEE PROFILE

Enhanced Design Pattern Definition Language

Salman Khwaja and Mohammad Alshayeb
Information and Computer Science Department
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia
e-mail: {khwaja & alshayeb} @kfupm.edu.sa

***Abstract*—Design patterns are abstract descriptions of object-oriented designs, which appear repeatedly for a possible high-quality solution. Many design pattern description languages have been proposed. These languages use a combination of a natural language, UML-style diagrams, complex mathematical or logic based formalisms, or eXtensible Markup Language (XML). In this paper, we propose an extension to the Design Pattern Description Language (DPDL), which is based on XML to support composite design patterns. A composite pattern is a special type of design patterns that is formed from a composition of other patterns. Composite patterns capture the synergy arising from the different roles an object plays in the overall composition structure. The enhanced Design Pattern Description Language (eDPDL) is found to be effective in capturing the composite design pattern while representing the whole composite design pattern in a single description.**

Keywords—design pattern language; composite design patterns; UML; XML; DPDL

I. INTRODUCTION

A composite design pattern (also called as composite pattern) is a special type of design pattern that represents a design theme, which keeps recurring in specific contexts. Composite design patterns are the composition of other simple design patterns. The main purpose of the composite design pattern is not to join multiple design patterns but it is to capture synergy in the overall structure of the system. Therefore, composite patterns are more than just the sum of the constituting patterns [1].

One of the purposes of the composite design patterns is to enable a higher level of reuse than individual design patterns and objects [2][3][4]. The modeling of the structure and behavior of the composite design patterns is usually done on object-oriented modeling techniques that use graphical notations such as the Unified Modeling Language (UML) [5][6]. UML has become one of the most widely used general-purpose languages for specifying, constructing, visualizing, and documenting artifacts of software-intensive systems. It provides a collection of notations to capture different aspects of the system and sub-systems under development [7].

The objective of this paper is to propose an extension to the Design Pattern Definition Language (DPDL) [8], which is called extended Design Pattern Definition Language (eDPDL), to be able to express the composite design patterns in a reusable fashion. DPDL was originally created to share design pattern implementation details. DPDL already covers the structural and behavioral aspects of the design pattern and is also flexible. However, DPDL is restricted to specify only the structural and behavioral aspects of a single design pattern. DPDL does not provide any means to specify that a particular component or action is originally part of some simple design pattern. Therefore, the composite design pattern description in DPDL becomes a description of one big complex design pattern instead of the aggregation of few simple design patterns.

This motivated us to propose enhancement to DPDL in order to handle composite design patterns. This will enable us to distinguish the components of individual design patterns and their behavior, which makes the composite design pattern less complex and more understandable.

The paper is organized as follows: Section 2 contains the literature review, in Section 3, the proposed enhancement is presented in Section 4. In Section 5, we present an example to validate the proposed enhancement and finally the conclusion is presented in Section 6.

II. LITERATURE REVIEW

In our literature survey, we could identify only three composite design patterns. These are: Active Bridge, Bureaucracy and Model View Controller (MVC) design patterns.

Active Bridge is usually used in recurring types of frameworks, where the application is needed to be connected with a resource like widget or inter-process communication channel. At the heart of the Active Bridge pattern is Bridge Pattern. Other than that proxy, Observer, Abstract Factory and Factory method design patterns are also used for different components of Active Bridge [9][10].

The second commonly mentioned composite design pattern is Bureaucracy. Bureaucracy design pattern is created using Chain of Responsibility, Composite, Mediator and Observer design patterns. Bureaucracy is also considered as a complex design pattern since it is used in the resource management and interaction of the complex objects. This pattern is highly efficient in developing large application where consistency is important [11]. This design pattern is used in many frameworks including ET++ [10], InterViews [12] and SmallTalk Framework [13].

The most commonly used composite design pattern is Model View Controller (MVC). MVC is also used in designing 3-tier or n-tier architecture frameworks. It is used to handle multiple user interfaces based on the user information or interaction. MVC allows modifying a user interface independent of the application logic or data associated with it [14]. It is usually based on Observer and Strategy design patterns. There are many variations of this design pattern used

 ISBN: 978-1-61208-367-4

in the industry like Model View Presenter (MVP) [15] and Model View ViewModel [16].

Although many researchers have tackled the problem of design pattern description or definition languages but very few worked on the language for the composite design pattern definition or description.

Vlissides proposed a visual notations called Pattern: Role annotation that adds scalability and readability over the Venn Diagram notation [17]. This notation focused on static properties of the design pattern compositions. The notation failed to capture the behavioral aspect of the operations in a design pattern.

Dong et al. [18] used First Order Logic (FOL) theories to specify the structural aspect of patterns and Temporal Logic of Action (TLA) of specify their behavioral aspect. The same techniques were used to specify pattern composition. The specification of the structural aspect of a pattern used predicates for describing classes, state variables, methods and their relations.

Dong et al. also investigated the commutability of pattern instantiation with pattern integration (composition). A pattern instantiation was defined as a mapping from names of various kinds of elements in the pattern to classes, attributes, methods, etc. in the instance. An integration of two patterns was defined as a mapping from the set union of the names of elements in the two patterns into the names of the elements in the resulting pattern. This formal definition of integration is mathematically equivalent to the multiple name mapping approach [18].

Taibi and Ngo [19] also took an approach very similar to the one by Dong et al. Instead of defining mappings for pattern compositions and instantiations, they used substitution to directly rename the variables that represents pattern elements. For instantiation, the variables are renamed to constants, whereas for composition, they are renamed to new variables. The composition of two patterns is then the logical conjunction of the predicates that specify the structural and behavioral properties of the patterns after substitution.

Helm et al. [20] used notion of contracts for describing the behavioral composition of the objects. However, his approach was much broad and not specific to composite design patterns. In addition, it only emphasized on the functional or behavioral aspect of the system and the interactions of the objects in the system.

All of these approaches could be used for composite design patterns but they were not specifically designed for the composite design patterns but were for general composition of design patterns in the system.

Riehle [21] investigated the composite design patterns as a recurring framework. In his technique, he used role-based analysis and described the design patterns composition using role-diagrams. Role-diagrams were supplemented with composition constraints, which specify the set of roles an object may, have to, or must not play.

Dong [22] studied the composite patterns in formal settings. He called composition of two or more patterns as name mapping. He defined name mapping as “classes and objects declared in a pattern with the classes and objects declared in the composition of this pattern and other patterns" [22]. Dong used formal mathematical specification for the structural and behavioral properties of the instance of the composite design pattern.

III. DESIGN PATTERN DESCRIPTION LANGUAGE

DPDL a design pattern description language that provides a flexible and a simple way to express patterns [8]. DPDL covers the maximum possible characteristics of the design pattern in a simple way. Figure 1 shows the high level schema for the DPDL language. At the left most in the diagram is the DesignPattern element; for each design pattern there is a DesignPattern element.

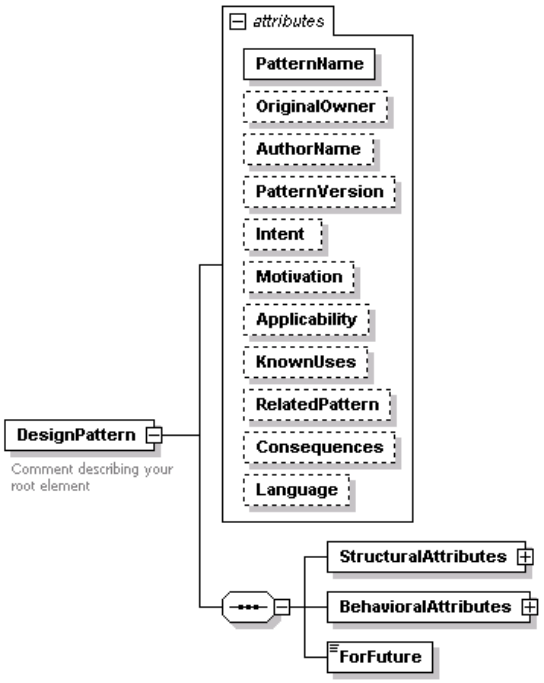

Figure 1. DPDL High Level Schema

As can be seen from Figure 1, the design pattern element has three sub elements; (a) structuralAttributes, (b) behavioralAttributes and, (c) ForFuture. The Structural attribute covers the structural properties of the design pattern. The behavioral attribute defines the behavioral aspect of the design pattern. Finally, ForFuture element is for extending DPDL to add other elements to cover new features of the design pattern in the future.

IV. THE PROPOSED ENHANCEMENTS

Enhancements are made on the original DPDL schema in order to handle the composite design patterns. This section covers these changes. The changes made on the DPDL to handle composite patterns are done on the attributes; no new elements were introduced. Therefore, eDPDL schema is backward compatible; thus, all the existing design pattern instances created using DPDL are still valid on eDPDL.

 ISBN: 978-1-61208-367-4

Two new attributes: isComposite and ConstituentPatterns have been added to DesignPattern element, as shown in Figure 2. *isComposite* attribute is of Yes/No type; if this attribute is Yes that means the description is for a composite pattern thus the designer of the pattern needs to put the design patterns involved in the composite pattern in the *ConstituentPatterns* attribute. *ConstituentPatterns* attribute is of a list type, which means that this attribute can have a list of values delimited by a space.

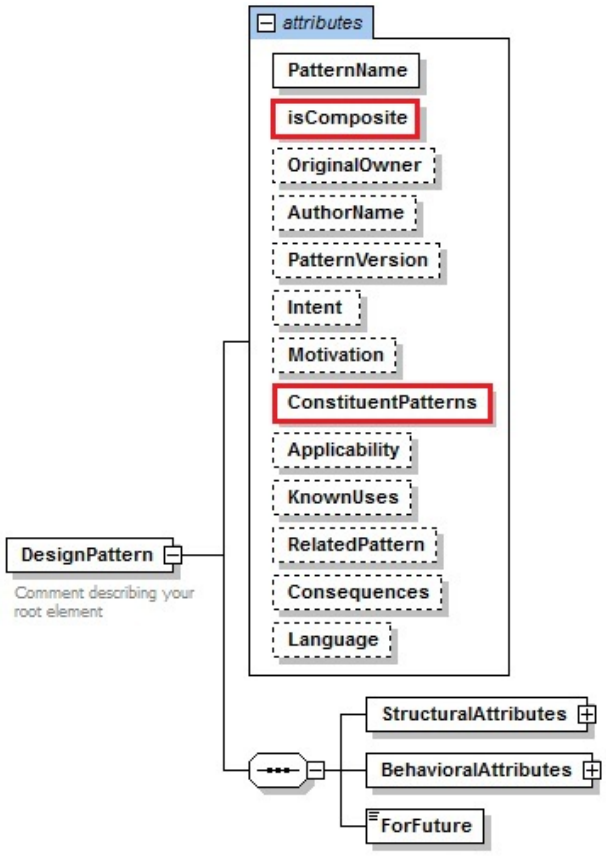

Figure 2. Changes in Attributes of DesignPattern element of DPDL

A. *Changes in StructuralAttribute's Elements*

In the StructuralAttributes element, there are four elements. These elements are Classes, Objects, Operations and Relationships. Each of these elements has a subgroup element. The changes made in StructuralAttributes element are restricted to the changes in the subgroup element of the four main elements of the StructuralAttribute's element. The changes are shown in Figure 3.

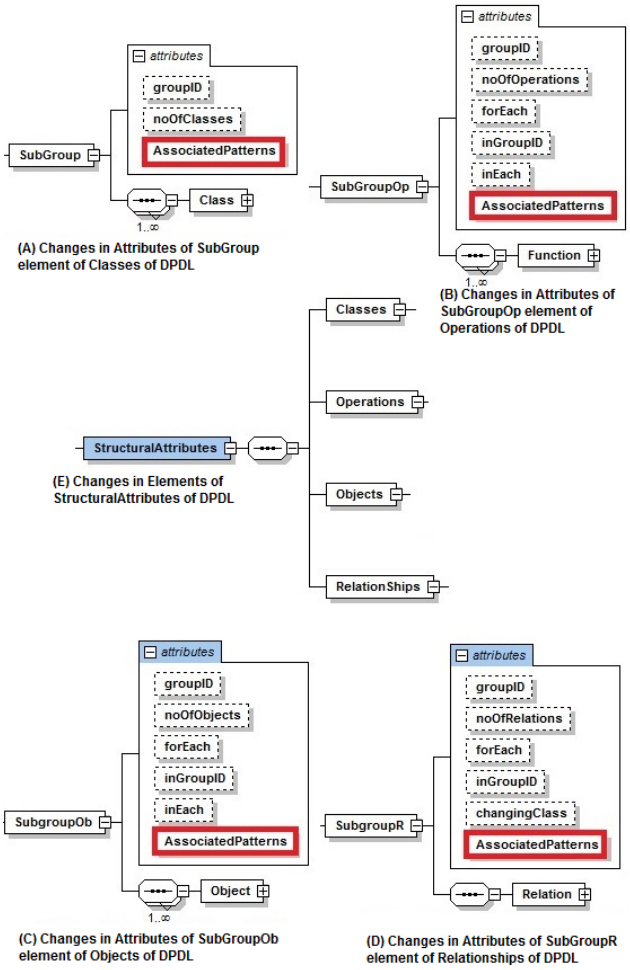

Figure 3. Structural Attributes of DPDL and changes made for eDPDL

As can be seen from Figure 3, AssociatedPatterns attribute (highlighted with a thick rectangle) has been added, hence, each element of class, operation, object or relationship is linked to one or more pattern of the composite design pattern. Therefore, an operation belonging to a particular design pattern in a composite design pattern is mentioned by giving the name of that particular design pattern in the AssociatedPatterns attribute for that particular subgroup element of the operation.

It is also important to mention that attribute name ("AssociatedPatterns") is used in the plural form. This means that multiple design patterns can be listed in this attribute. These patterns can be listed using space delimited. This is done because in some cases a class in a composite pattern might be represent two different patterns in a single composite design pattern.

 ISBN: 978-1-61208-367-4

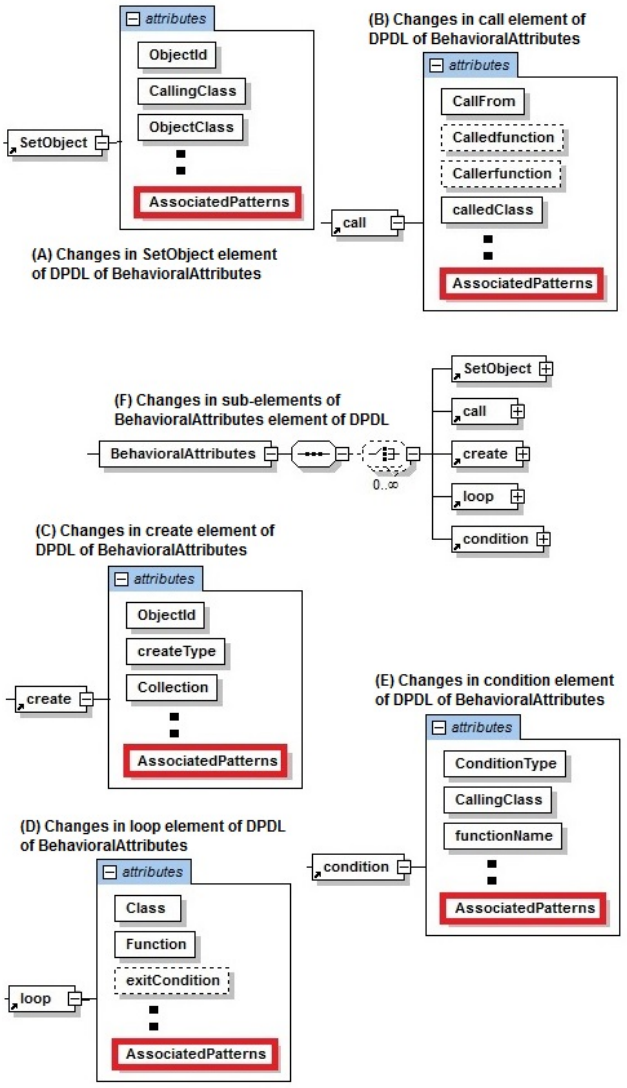

Figure 4. Behavioral Attributes of DPDL and changes made for eDPDL

B. *Changes in BehavioralAttribute's Elements*

The second part of DPDL language is the Behavioral Attributes element. This element has five sub elements describing the behavioral aspect of the design pattern. In all elements related to the behavioral attributes of the design pattern an AssociatedPatterns attribute is added. The changes made for the eDPDL in the DPDL are shown in the Figure 4.

V. EDPDL VALIDATION

Model View Controller (MVC) is a software architecture pattern, which separates the representation of information from the users' interaction with it. There are three types of objects in MVC. Application data is represented by Model, the View is the output or the screen shown to the user, and the Controller handles all the reaction to the input. The Publish-Subscribe protocol is used between model and view - when Model data is changed it will update the View. It also allows attaching multiple Views to the same Model. This is achieved by using the Observer design pattern [23]. Controller implements a particular Strategy for the View, which is similar to the Strategy design pattern. Therefore, this makes MVC a composite design pattern with two design patterns Observer and Strategy. There are different variations of the Model View Controller (MVC) design pattern. Below is one of them [2].

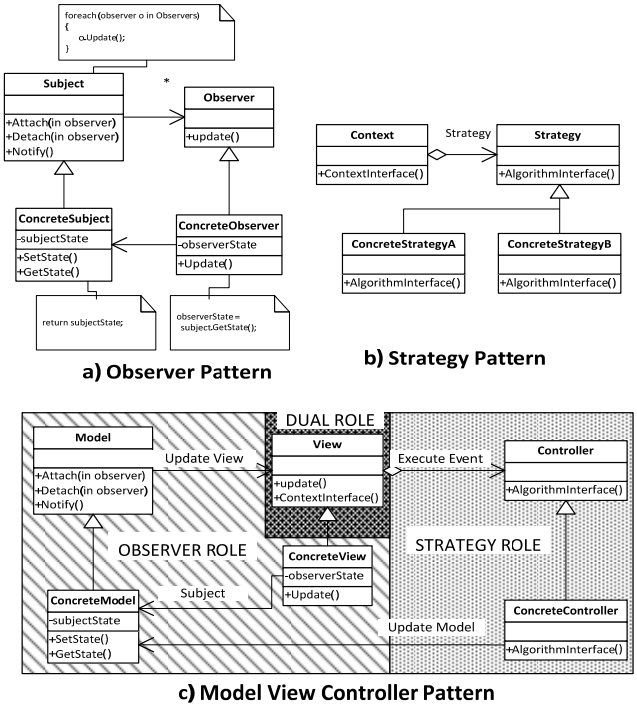

Figure 5. Model View Controller Class Diagram

As can be seen in Figure 5 the shown version of the Model View Controller (MVC) design pattern is composed of Observer and Strategy patterns. The Observer pattern is shown on the left side and the Strategy pattern is on the right side. The view class performs the role of both Strategy design pattern and observer design pattern.

This example shows that there can be a component in the composite design pattern, which acts for more than one design pattern. The update operation in the View class of Model View Controller design pattern is acting in a role of Observer and the contextInterface operation is acting in a role of Strategy design pattern.

We can see in Classes Node that Model Class is defined as part of Observer design pattern and similarly Controller class is defined as link to the Strategy design pattern. However, View Class is shown as part of both Observer and Strategy design group.

Similarly, in Operations group, different operations are also linked with their respective design pattern by listing the pattern in the AssociatedPattern element of the particular operation subgroup. Similarly, same approach is used in the Objects and Relationships Nodes.

 ISBN: 978-1-61208-367-4

```
<StructuralAttributes>
        <Classes>
        <SubGroup groupID="ModelClassGroup" noOfClasses="1" >
        <Class className="Model" isAbstract="Yes" isParent="Yes"
hasConstructor="Yes" classModifier="public" isDerived="No"/>
        </SubGroup>
        <SubGroup groupID="ConcenteModelClassGroup"
noOfClasses="1" >
        <Class className="ConcreteModel" isAbstract="No"
isParent="No" hasConstructor="Yes" classModifier="public"
isDerived="Yes" parentId="Model"/>
        </SubGroup>
        <SubGroup groupID="ViewClassGroup" noOfClasses="1">
        <Class className="View" isAbstract="Yes" isParent="Yes"
hasConstructor="Yes" classModifier="public" isDerived="No" />
        </SubGroup>
        </Classes>
        <Operations>
        <SubGroupOp>
        <Function returnType="Null" containingClassId="Model"
functionName="Attach" functionModifier="public"
inputVariablesType="View" />
        </SubGroupOp>
        </Operations>
        <Objects>
        <SubgroupOb>
        <Object objectName="Views" objectClass="ICollection"
containingClass="Model" objectModifier="private" isList="Yes"
ListType="ICollections"/>
        </Objects>
        <RelationShips>
        <SubgroupR>
        <Relation endClass="View" initiatingClass="ConcreteView"
relationName="Generalization" />
        </SubgroupR>        </RelationShips>
</StructuralAttributes>
```

Figure 6. Example of Structure Attributes of MVC Design Pattern written in DPDL

```
<StructuralAttributes>
        <Classes>
            <SubGroup groupID="ModelClassGroup"
noOfClasses="1" AssociatedPatterns="Observer">
                <Class className="Model" isAbstract="Yes"
isParent="Yes" hasConstructor="Yes" classModifier="public"
isDerived="No"/>
            </SubGroup>
            <SubGroup groupID="ViewClassGroup"
noOfClasses="1" AssociatedPatterns="Observer Strategy">
                <Class className="View" isAbstract="Yes"
isParent="Yes" hasConstructor="Yes" classModifier="public"
isDerived="No" />
            </SubGroup>
            <SubGroup groupID="ControllerClassGroup"
noOfClasses="1" AssociatedPatterns="Strategy">
                <Class className="Controller" isAbstract="Yes"
isParent="Yes" hasConstructor="No" classModifier="public"
isDerived="No"/>
            </SubGroup>
        </Classes>
        <Operations>
            <SubGroupOp AssociatedPatterns="Strategy">
                <Function returnType="Null"
containingClassId="View" functionName="ContextInterface"
functionModifier="public" inputVariablesType="Null" />
            </SubGroupOp>
        </Operations>
        <Objects>
            <SubgroupOb AssociatedPatterns="Observer">
                <Object objectName="Model"
objectClass="ConcreteModel" containingClass="ConcreteModel"
objectModifier="private"/>
            </SubgroupOb>
        </Objects>
        <RelationShips>
            <SubgroupR AssociatedPatterns="Observer">
                <Relation endClass="Model"
initiatingClass="ConcreteModel"
relationName="Generalization"></Relation>
            </SubgroupR>
        </RelationShips>
</StructuralAttributes>
```

Figure 7. Example of Structure Attributes of MVC Design Pattern Written in eDPDL

```
    <BehavioralAttributes>
        <create ObjectId="views" callingClass="Model"
returns="null" Collection="Yes" objectClass="ICollection"
createType="ReadOnly" AssociatedPatterns="Observer"/>
        <call callingClass="Model" returns="null"
CallFrom="function" variableType="{Views}" calledClass="View"
variables="{v}" Callerfunction="Attach" Calledfunction="Add"
AssociatedPatterns="Observer" />
        <call callingClass="Model" returns="null"
CallFrom="function" variableType="{Views}" calledClass="View"
variables="{v}" Callerfunction="Detach" Calledfunction="Remove"
AssociatedPatterns="Observer" />
        <call callingClass="Model" returns="null"
CallFrom="function" variableType="{Views}" calledClass="View"
variables="{v}" Callerfunction="Notify" Calledfunction="Update"
AssociatedPatterns="Observer" />
        <create ObjectId="Controller" callingClass="View"
returns="null" Collection="null" objectClass="Controller"
createType="Readonly" AssociatedPatterns="Observer" />
        <SetObject CallingClass="View" ObjectId="controller"
ObjectClass="Controller" SetTo="Controller"
AssociatedPatterns="Observer" />
        <call callingClass="View" returns="null"
CallFrom="Function" variableType="null"
calledClass="ConcreteController" variables="null"
Calledfunction="AlgorithmInterface" calledThrough="Controller"
Callerfunction="ContextInterface" AssociatedPatterns="Observer" />
        <create ObjectId="ViewState" callingClass="object"
returns="null" Collection="No" objectClass="object"
createType="null" AssociatedPatterns="Observer" />
        <SetObject CallingClass="ConcreteView" ObjectId="Model"
ObjectClass="ConcreteModel" SetTo="Model.ModelState"
AssociatedPatterns="Observer" />
        </BehavioralAttributes>
```

Figure 8. Example of Behavioral Attributes of Model View Controller Design Pattern

Figure 6 shows the structural view of the MVC design pattern written using the original definition of DPDL. Figure 7 and Figure 8 show the structural and the behavioural views of the MVC Design Pattern written in eDPDL respectively. As can be seen, the structural view of the original DPDP does not have AssociatedPatterns tag. Without having this tage, it will be impossible to recognize if the described design pattern is a one large design pattern or the sum of two or more design patterns.

The eDPDL is an extension of DPDL, which not only handles describing the regular single design patterns in a singular fashion but can also describe composite design

 ISBN: 978-1-61208-367-4

pattern as a combination of two or more design patterns. This makes it clear if the design pattern is a composite design pattern or not. The original DPDL description cannot differentiate between composite design patterns and single design pattern. eDPDL is also backward compatible, that is all design pattern which were described based on DPDL schema will work on the schema of eDPDL without any change.

VI. CONCLUSION AND FUTURE WORK

Composite design patterns are usually handled through UML or formal mathematical notations, which are either too complicated or they do not cover the roles and operations comprehensively for the composite design patterns. Thus, the roles that the classes, operations, and attributes play in the pattern get lost. To accomplish the goals of the design pattern, pattern related information becomes important. If this information is not explicitly, the designers are forced to communicate at the class and object level, instead of the pattern level [24].

In this paper, we proposed an extension to DPDL to handle the composite design patterns. The proposed extension, eDPDL, adds attribute to DPDL to handle composite patterns in an easy and efficient way. An example was provided and we found that eDPDL is effective in handling composite design patterns and is also easily understandable as it is built on XML.

Our future research includes extending eDPDL to include other design patterns such as security. We also plan to provide an automated tool to fully support eDPDL.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum and Minerals, Saudi Arabia.

REFERENCES

[1] D. Riehle and H. Züllighoven, “Understanding and using patterns in software development,” Theor. Pract. Object Syst., vol. 2, 1996, pp. 3-13.

[2] A. Shelest. Model View Controller, Model View Presenter, and Model View ViewModel Design Patterns. Available: http://www.codeproject.com/Articles/42830/Model-View-Controller-Model-View-Presenter-and-Mod. [Retrieved 2014: 18 August 2014].

[3] P. Alencar, D. Cowan, J. Dong, and C. Lucena, “A pattern-based approach to structural design composition,” in Computer Software and Applications Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International, 1999, pp. 160-165.

[4] J. Dong, “Design component contracts: model and analysis of pattern-based composition,” Ph. D. Thesis, Computer Science Department, University of Waterloo, 2002.

[5] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language User Guide," ed: Addison-Wesley, 1999.

[6] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual vol. 2: Addison-Wesley, 2005.

[7] L. Fuentes-Fernahndez and A. Vallecillo-Moreno, “An Introduction to UML Profiles,” in The European Journal for the Informatics Professional, 2004, pp. 5–13.

[8] S. Khwaja and M. Alshayeb, “Towards design pattern definition language,” Software: Practice and Experience, vol. 43, 2013, pp. 747–757.

[9] P. M. Yelland, “Creating host compliance in a portable framework: a study in the reuse of design patterns,” ACM SIGPLAN Notices, vol. 31, 1996, pp. 18-29.

[10] A. Weinand and E. Gamma, “ET++–a portable, homogenous class library and application framework,” Computer Science Research at UBILAB, 1994, pp. 66-92.

[11] D. Riehle, “Bureaucracy,” 1997,

[12] M. A. Linton, J. M. Vlissides, and P. R. Calder, “Composing user interfaces with InterViews,” Computer, vol. 22, 1989, pp. 8-22.

[13] D. Riehle, B. Schäffer, and M. Schnyder, “Design of a Smalltalk Framework for the Tools and Materials Metaphor,” Informatik/Informatique, vol. 3, 1996, pp. 20-22.

[14] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern Oriented Software Architecture: On Patterns and Pattern Languages vol. 6: Wiley, 2007.

[15] M. Potel, “MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java,” Taligent Inc, 1996,

[16] J. Smith, “WPF apps with the model-view-ViewModel design pattern,” MSDN magazine, 2009,

[17] J. Vlissides, "Notation, Notation, Notation. C++ Report," 1998.

[18] J. Dong, T. Peng, and Y. Zhao, “On Instantiation and Integration Commutability of Design Pattern,” The Computer Journal, vol. 54, 2011, pp. 164-184.

[19] T. Taibi and D. C. Ngo, “Formal specification of design pattern combination using BPSL,” Information and Software Technology, vol. 45, March 2003, pp. 157-170.

[20] R. Helm, I. M. Holland, and D. Gangopadhyay, “Contracts: specifying behavioral compositions in object-oriented systems,” in European conference on object-oriented programming on Object-oriented programming systems, languages, and applications, 1990, pp. 169-180

[21] D. Riehle, “Composite design patterns,” 1997, pp. 218-228.

[22] J. Dong, P. S. Alencar, and D. D. Cowan, “Ensuring structure and behavior correctness in design composition,” in Engineering of Computer Based Systems, 2000.(ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshopon the, 2000, pp. 279-287.

[23] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software: Addison Wesley, 1994.

[24] J. Dong, “UML Extensions for Design Pattern Compositions,” Journal of Object Technology, vol. 1, 2002, pp. 151-163.

 ISBN: 978-1-61208-367-4