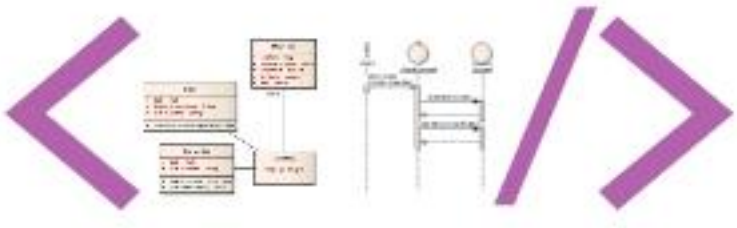

DESIGN PATTERN DEFINITION LANGUAGE (DPDL)

USER GUIDE

VERSION 1.0

DATE: DECEMBER 13, 2010

TABLE OF CONTENTS

LIST OF FIGURES

DPDL SCHEMA

This document presents the description of all DPDL schema (version 1.0) elements.

1. DESIGN PATTERN ATTRIBUTES

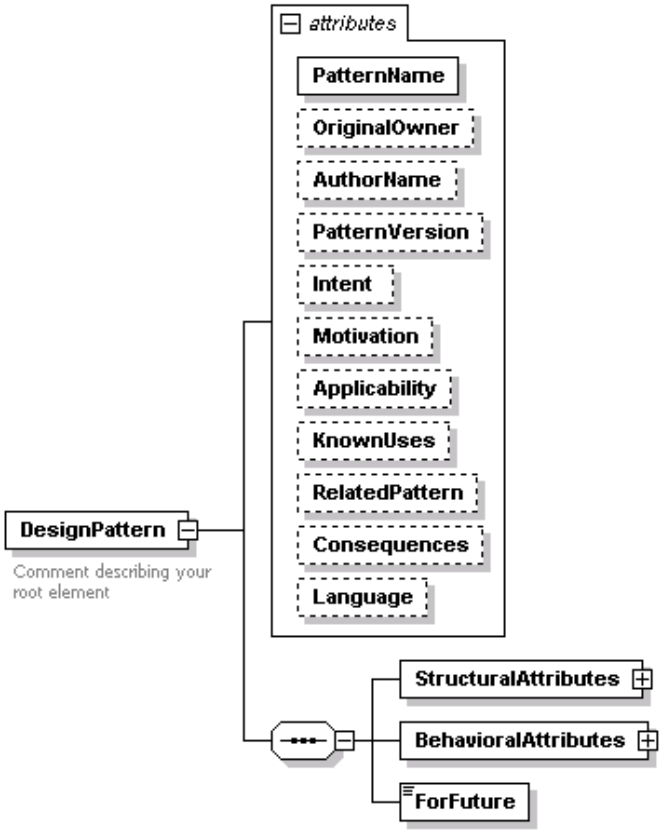

Figure 1: DPDL High Level Schema

Design Pattern attributes for the DPDL are as follows:

1.1 Pattern Name (Mandatory)

Pattern Name is the mandatory element of the DPDL *DesignPattern* attributes. The design pattern name is a handle which we use to describe a design problem, its solutions, and consequences in a word or two. Also naming a design pattern immediately increases our design pattern vocabulary. It makes it easier to think about designs and to communicate them and their trade-offs to others. Finding good names is an important task for design pattern developers.

1.2 Owner Name

The owner name identifies the person, who originally introduced this design pattern.

1.3 Author Name

The author name attribute indicates the name of the author who is designing this design pattern. He can be the head of the software team or the architecture designer or any researcher who is proposing a new design pattern.

1.4 Design Pattern Version

With the passage of time many original design patterns have got different version providing more specialized capabilities. So a simple design pattern name is not enough in some cases. Therefore with the help of design pattern version we can more accurately identify the design pattern.

1.5 Intent

Intent is a short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address? Intent is a textual field.

1.6 Motivation

Motivation is a scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem. The scenario will help one understand more about the pattern that follows.

1.7 Applicability

Applicability answers the question like what are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations? It is also a textual field in DPDL.

1.8 KnownUses

This field is added to provide some examples of this design pattern found in practical applications and real systems.

1.9 Related Patterns

This field answers the questions like which design patterns are closely related to this one.

1.10 Consequences

The consequences are the results and trade-offs of applying the design pattern. The consequences are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern. The consequences for software often concern space and time trade-offs. They may address language and implementation issues as well. Since reuse is often a factor in object-oriented design, the consequences of a pattern include its impact on a system's flexibility, extensibility, or portability. Listing these consequences explicitly helps you understand and evaluate them.

1.11 Language

If this design pattern is created for some application, then the language of the application can be mentioned in this attribute. The graphical output tool can also use this attribute to output correct diagram for the design pattern based on the language of the design pattern.

2. STRUCTURAL ATTRIBUTES

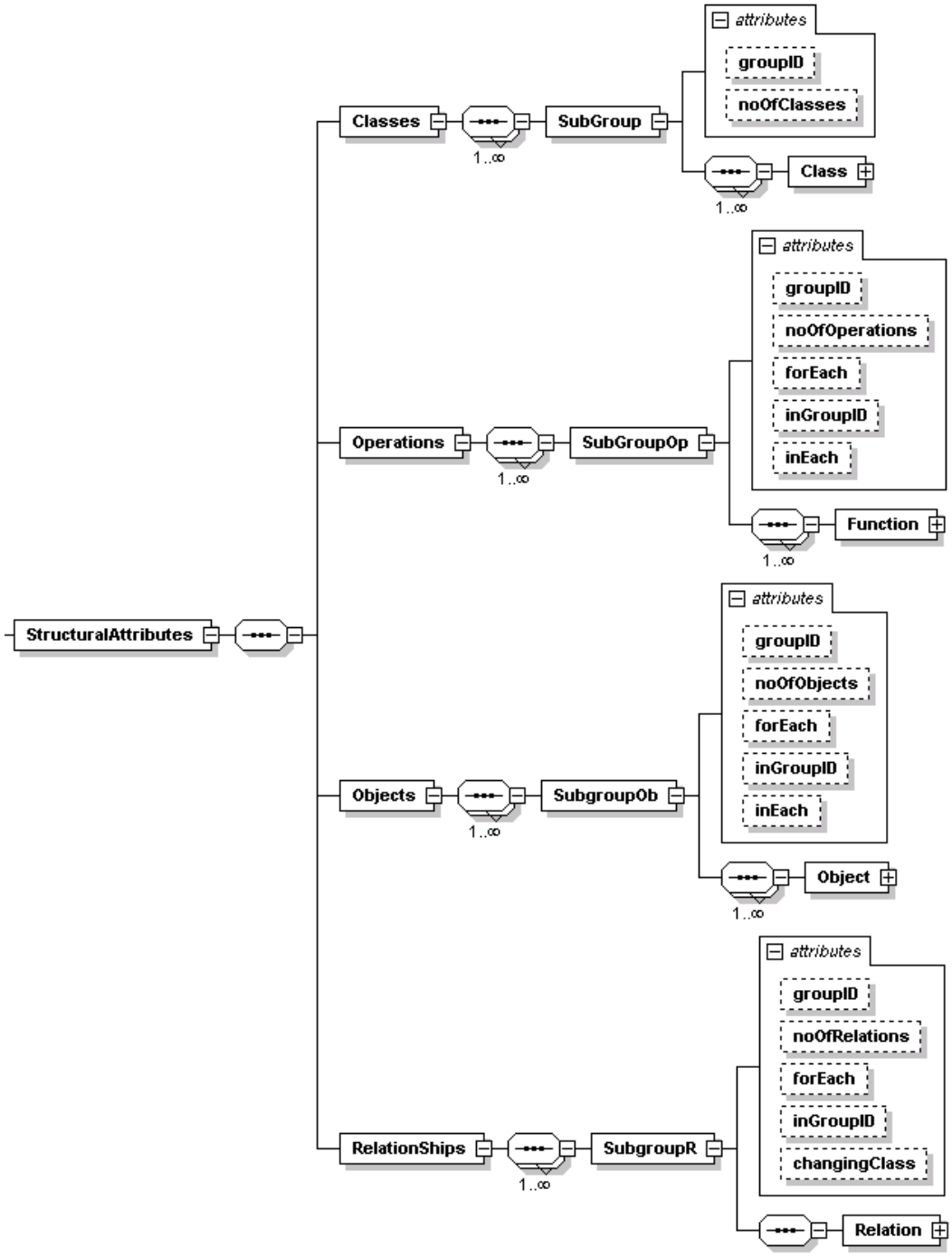

Figure 2: DPDL's Structural Attributes

2.1 SubGroup Element

The purpose of subgroup inside each structural element of DPDL is to help in making a template of a design pattern. For example a single instance of a particular design pattern may have 2 children of a particular class and another instance of the same design pattern may contain 5 children of a particular class. Both design patterns are of the same type. So to verify these two different variations of the same design pattern we should have a single template for that design pattern, so that, all of the variations of the design pattern are handled in a clear and concise way. One way is that each

possible instance of a design pattern gets its own schema. This means that almost infinite schemas of the template will be created handling each instance which is not feasible at all. Also if in future some changes are made in a design pattern schema then these changes are required to be repeated in all the templates that were created for that design pattern.

In our case we have introduced a SubGroup element in the four main elements (classes, operations, objects and relationships) of the structural attributes of DPDL. The attributes of subgourp elements are used for creating a very generic design pattern template which can cover many different scenarios.

The detail description of the four elements in the structural attribute element is given in the following sections.

2.2 Classes Element

The classes element is a group of class element. All the participant classes of the design patterns are going to be mentioned in the classes element of the structural attributes. The subgroup element's attributes are mostly used for describing the template of a design pattern. A template designed in DPDL will able to handle all the possible instances of that particular design pattern. As the subgroup elements attributes are extending the group it contains (in the case of *Classes Element* subgroup is extending class elements), so for understanding the subgroup attributes we need to know *class element's* attributes first. Therefore the description and explanation of subgroup element's attribute is given after the class element's attributes.

2.3 Class Element

The class element is used in describing a class in a design pattern. All the details about the class of a design pattern will be mentioned in the class element. It has its sets of attributes that will help to describe the class.

2.3.1 Class Element Attributes

The attributes of class are described below. The attributes are shown in Figure 3

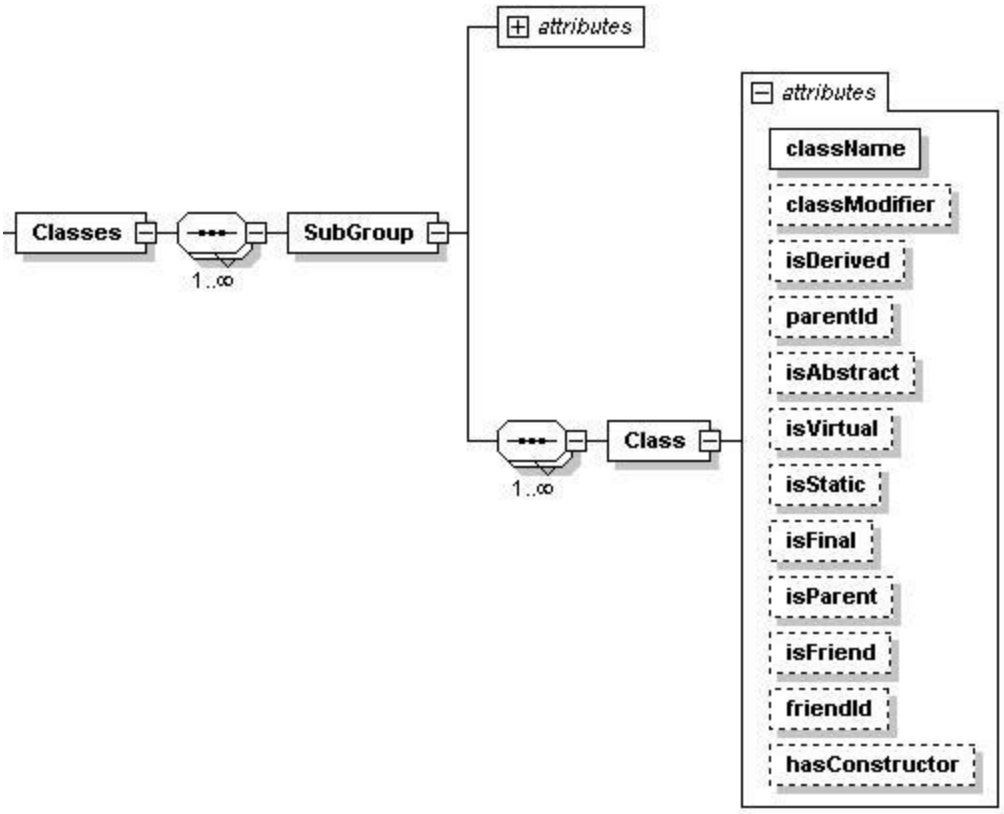

Figure 3: Attributes of Class Element of DPDL

2.3.1.1 ClassName (Mandatory)

ClassName is the most important attribute for the class element. It uniquely identifies the class in the design pattern. Each ClassName should be distinct in a particular design pattern.

2.3.1.2 ClassModifier (Optional)

The attribute ClassModifier identifies if the class is private, public or protected. It can only have one of the predefined values, so that the user does not insert a wrong value for this attribute.

2.3.1.3 isDerived (Optional)

This attributes is Yes for those classes which are derived from some other class. Otherwise the value of isDervived is no.

2.3.1.4 ParentId (Optional)

If a class is a derived class then the class id of the parent class can be written in here. This attribute is more for the validation, otherwise in creating a graphical output or defining design pattern, it is not mandatory.

2.3.1.5 isAbstract (Optional)

This attribute is Yes for abstract classes for other classes the value of isAbstract is no.

2.3.1.6 isVirtual (Optional)

The attribute isVirtual is Yes for virtual classes for other classes the value of isVirtual is no.

2.3.1.7 isStatic (Optional)

The classes which are static will have isStatic value as Yes and for other classes the value of isStatic is No.

2.3.1.8 isFinal (Optional)

The classes which cannot be used as a base class for any other class are known as non-extendable class. Final is the keyword used for them in java and in C# "sealed" keyword is used for such classes. These classes are in DPDL represented by setting the attribute isFinal's value to Yes. Default value for isFinal is No.

2.3.1.9 isFriend (Optional)

Friend classes are those classes which can access other classes methods and attributes without being related directly. These classes in our design pattern language (DPDL) are represented by having attribute isFriend set as Yes.

2.3.1.10 friendId (Optional)

If a class is a friend class then the id of the friend class will be declared as this attributes value.

2.3.1.11 hasConstructor (Optional)

If a class has one or more than one constructor then this value will be Yes, otherwise hasConstructor will have a value of No. The details about each constructor will be given in the Function element of the DPDL.

2.3.1.12 isParent (Optional)

Those classes which are parent to some other class or base class for other classes will have this attribute value set to Yes. Other classes which are not parent will have the value as No

2.3.2 SubGroup Element Attributes

The major use of subgroup element is in the template. Therefore its attributes are optional as the instance of design pattern can also be created in DPDL without using these attributes.

2.3.2.1 GroupId (Optional)

The unique id of any group will be mentioned in this attribute. If one part of design pattern (a part can be a structural attribute like class, object, relationship and function) is dependent on another part of the design pattern then the group id of the independent part will be referenced in the dependent part. For example if there is one function against each class of a particular group, then the subgroup of the function, will refer the *groupId* of the class on which the function group is dependent. Class group will be independent in this example and its *groupId* will be used in the function subgroup.

2.3.2.2 noOfClasses; (Optional)

This attribute defines how many instances of the class in this group have, which are exactly like the class defined in this subgroup through its attribute. This attribute can have numeric as well as textual value. So we can have values like 1, 2 or 5 etc, also a value like "*one-to-many*" is acceptable, which can be used in defining templates. For

example, suppose there is a class in the group, which is inherited from Shape class, then if we have value of noOfClasses as 3, then this means that there will be 3 classes in the design pattern, inherited from the Shape class. So in actual realization of the design pattern there will be three classes with all attributes of the class identical to the attributes mentioned for the class in that group except for the *className* attribute which has to be unique in code. This way by defining the attributes of only one class in the DPDL and setting noOfClasse to 3, we can represent and then create three classes with same attributes.

2.4 Operations Element

This part of the DPDL schema handles all the operations which are present in the design pattern. The sub-element of Operations is subGroupOp which is included for the purpose of handling template for design patterns. In the case of template of a design pattern, a function with same signature may be repeated in all classes of a particular group. Such situations can be handled by subGroupOp by describing just one operation in DPDL of the design pattern. Further detail about the subGroupOp attributes is given after the Function element, as subGroupOp attributes are extending the Function element's attributes.

In case of defining a particular instance of a design pattern each function may be described in a separate subGroupOp or all the functions can be described in a single subGroupOp. All the functions are the child element of SubGroupOp which is the child element of Operation. Using this hierarchy helps in creating a very simple, extendible and easily understandable hierarchy for grouping all the operations.
The Operations Element is the big container containing all the functions and operation of a particular design pattern inside it.

2.5 Function Element

The actual function details are contained in this element. Figure 4 shows the attributes of function element graphically

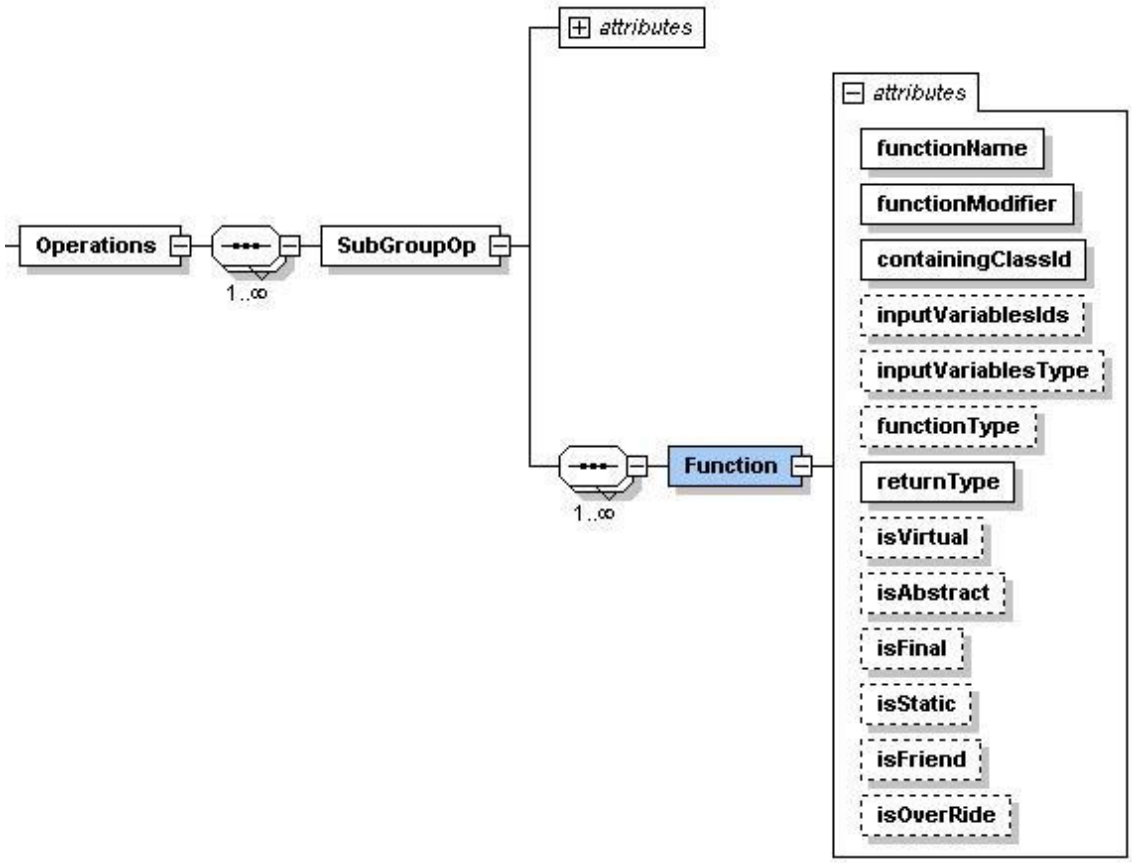

Figure 4: Attributes of Function Element in DPDL

2.5.1 Function element attributes

Following sub sections describe all the attributes of *Function Element*.

2.5.1.1 functionName (Mandatory)

The name of the function, method, operation, property which this element is defining, is given in the *functionName* attribute. This function name can also be used in code. In remaining attributes of this section, the word function covers operation, method or property.

2.5.1.2 functionModifier (Mandatory)

The modifier of the function is described in this attribute. Like classModifier the functionModifier also have pre-defined values from public, protect and private. This helps to avoid mistakes on the part of the designer and also helps with consistency.

2.5.1.3 containingClassId (Mandatory)

The id of the class in which the function is contained is defined in this attribute. As we are focusing on object oriented design pattern, therefore all functions should belong to some specific class.

2.5.1.4 inputVariablesId (Optional)

The variable name of all the variables which are the argument of the function are mentioned in this attribute in curly brackets '{}'. A comma as a separator is used

between two variable ids. In case there is no input argument to the function, then ‘null’ without curly brackets is used.

2.5.1.5 inputVariablesType (Optional)

The attribute inputVariableType stores the data type of the variable which are used as the argument of the function. They are also inside curly brackets ‘{}’. A comma as a separator is used between two variable types. The first inputVariablesType belongs to the first inputVariablesId and so on. This also means that the number of inputVariablesType should be equal to the number of inputVariablesId.
In case there is no input argument to the function, then ‘null’ without curly brackets is used.

2.5.1.6 functionType(Optional)

The attribute functionType tells what type of function it is. It is also a variable with pre-defined values of Method, Constructor, Destructor, Event, GetProperty and SetProperty. Default value is taken as method.

2.5.1.7 returnType (Mandatory)

This property tells the return type of the function. The return type can be integer, string or other data type or it can also be void, if there is no return type.

2.5.1.8 isVirtual (Optional)

If a function is a virtual function then this attribute is used to describe it. The value is Yes in the case of virtual function and No otherwise. Default value is No.

2.5.1.9 isAbstract (Optional)

This attribute is Yes for abstract functions for other functions the value of isAbstract is no. The default value is No.

2.5.1.10 isFinal (Optional)

If a function cannot be extended anymore then this property of isFinal is set to Yes. In other cases the value of isFinal is No. The default value is No.

2.5.1.11 isStatic (Optional)

For static functions the value of isStatic is Yes. When the function is not static then the value is No. Default value is No.

2.5.1.12 isFriend (Optional)

If a function can be accessed by other classes which are not the child class then the function is made as a friendly function. For such functions isFriend is set to Yes. Otherwise the value is No. Default value is No.

2.5.1.13 isOverRide

If the function is an overridden on a base class function then this property of the function is set to yes, in other cases it is No. Default value is No.

2.5.2 SubGroupOp Element Attributes

The major use of subgroupOp element is also in the template. Therefore its attributes are optional as the instance of design pattern can also be created in DPDL without using these attributes.

2.5.2.1 GroupId (Optional)

The unique id of any group is mentioned in this attribute. With this unique group id this group can be referenced in any part of the DPDL.

2.5.2.2 noOfOperations (Optional)

This attribute defines how many instance of similar operations are in the final instance of the design pattern. This attribute can have numeric as well as textual value. So we can have values like 1, 3 or 5 etc, also a value like "*one-to-many*" is acceptable, which can be used in defining templates.

2.5.2.3 inGroupId (Optional)

inGroupId is the group Id of another structural part of DPDL, which is referenced by this subgroup. It is used when number of operation in subgroup is dependent on another group of DPDL's structural part, then this attribute is used to identify the independent group. The inGroupId always come when forEach or inEach attribute is used.

2.5.2.4 forEach (Optional)

The value of this attribute can be class, object, operation, function. To understand the use of *forEach*, take an example of a design pattern template in which there is a separate set function for all the classes of a particular subgroup, child of classes element, in some class (say classA),. As discussed earlier, the template should handle all variation of design pattern, so different variation of the same design pattern can have different number of classes. So in template we have to say that for each class in *subGroup x*(where *x* is id group of independent subGroup), there should be a function in classA, with same input and return types for all classes of *subGroup x*. *forEach* attribute identifies for which structural part this function is repeated for. In our case it is class for which it is repeated for. Now the inGroupId identifies the id of the group (like *subGroup x*) whose number of classes it will be based on.

When *forEach* is used in the *subGroupOp* then only the *operationName* is changed. The numbers of identical function which are created are equal to the number of classes in the subgroup identified in *inGroupId* and all these functions have same containing class. It is also explained through figure in Figure 5.

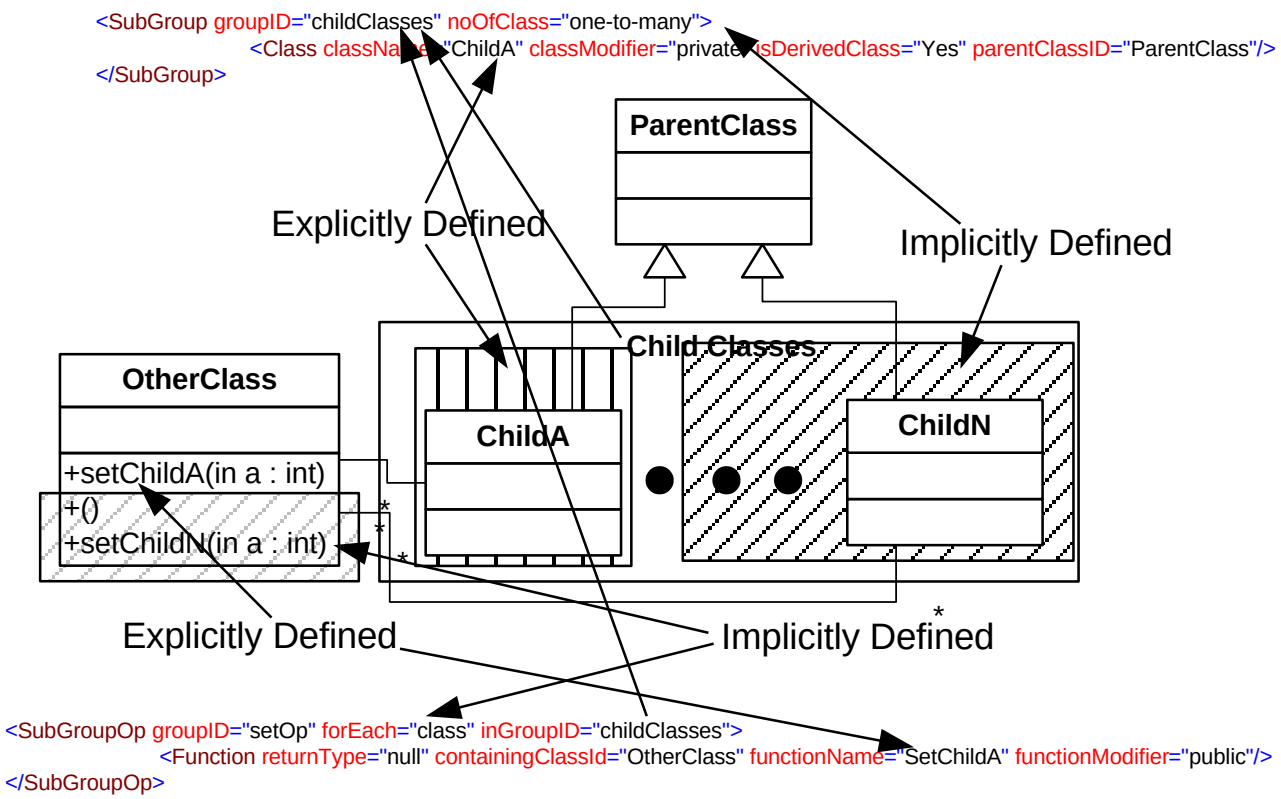

Figure 5: Example of forEach in Function.

2.5.2.5 inEach (Optional)

This variable is also used in conjunction with the *groupId* attribute. Whenever *inEach* is used in any type of subgroup, then there should be an *inGroupId* attribute present. *inEach* attribute is added to handle the situation when user wants to describe that a particular function is present in all the classes in a subgroup, and the value *noOfClasses* of that subgroup is more than 1. There can be two cases, one scenario is that we have some numeric value (e.g. 2), in *noOfClasses* of subGroup element. In this case we can either show two functions, one for each class, in the DPDL, or we can show it in the DPDL by just showing one function and have the value of *inEach* attribute as class and give the id of *subGroup* to *inGroupId*. This way it tells the programmer that exact function which is defined in *subGroupOp* is present in each class of a particular subgroup whose value is given in *inGroupId*. The value of the attribute *inEach* is class as it is referring to a subgroup which is child of classes element.

Second case is when we are defining a design pattern and the value of *noOfClasses* of subGroup element of classes, is "*one-to-many*". In this case we don't know the exact number of classes. So we require a way to mention that this function is dependent on a particular subGroup of Classes element, and each class of that subGroup needs to have this function. The id of that subgroup is refers in *inGroupId* attribute. It is shown in Figure 6.

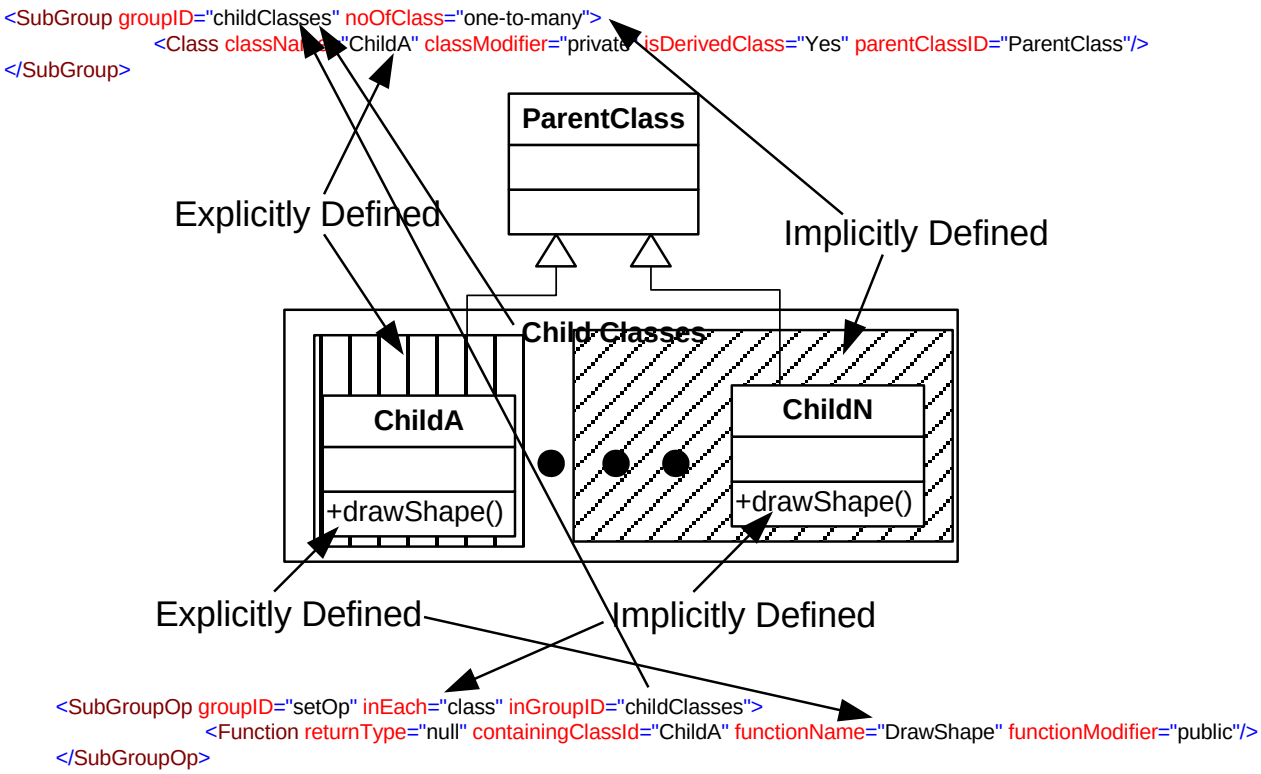

Figure 6: Example of inEach for Function

2.6 Objects Element

This element acts as a container for all the objects of the design pattern. The sub element of Object is SubGroupOb. SubGroupOb has the same purpose of providing support for the template design patterns by describing multiple objects through defining only one object of that type in the DPDL. The attributes of the SubGroupOb and their description is given after the Object Element.

2.7 Object Element

A single object is defined by the Object Element. All the attributes of a particular object are described in the object element.

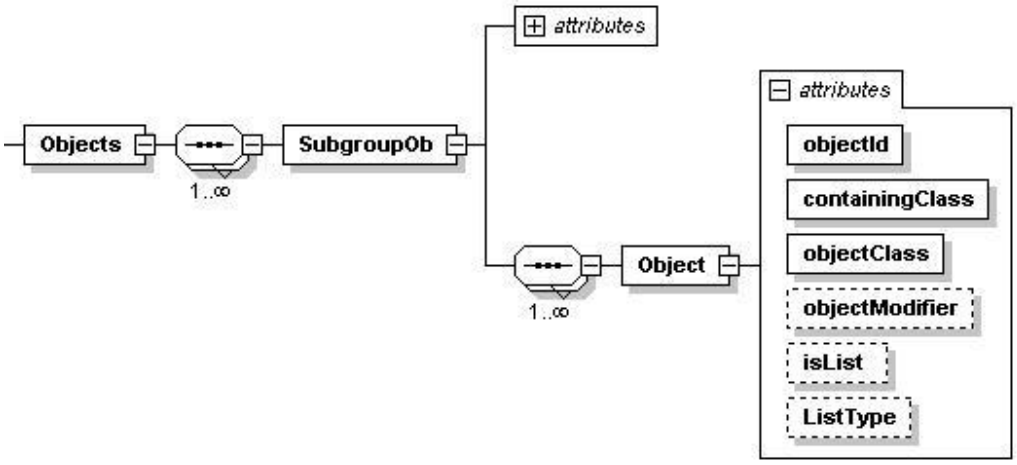

Figure 7: Attributes of Object Element in DPDL

2.7.1 Object Element Attribute

Following are the attribute which an object or a variable can have in DPDL.

2.7.1.1 objectName (Mandatory)

The objectName is the unique identifier for the object or the variable. An object can have same name if they are in two different classes

2.7.1.2 containingClass (Mandatory)

The containingClass attribute tells in which class the object is present.

2.7.1.3 objectClass (Mandatory)

The objectClass tells from which class the object belongs.

2.7.1.4 objectModifier (Optional)

The objectModifier like functionModifier tells the previlage level of the object. It also has predefined values of private, public or protected. Default value is private.

2.7.1.5 isList (Optional)

If the object is a list or an array then this attribute of the object is set to Yes, otherwise it is set to No. Default value is No.

2.7.1.6 ListType (Optional)

If the isList attribute is Yes, then ListType can be set to some array type like array, hash table or link list.

2.7.2 SubGroupOb Element Attributes

The major use of subgroupOb element is also in the template. Therefore its attributes are optional as the instance of design pattern can also be created in DPDL without using these attributes.

2.7.2.1 GroupId (Optional)

The unique id of any group will be mentioned in this attribute. With this unique group id this group can be referenced in any part of the DPDL and dependency between one part of the DPDL to another part can be created.

2.7.2.2 noOfObjects (Optional)

This attribute defines how many identical objects are there like the ones mentioned in this group. This attribute can have numeric as well as textual value. So we can have values like 1, 3 or 5 etc, also a value like "*one-to-many*" is acceptable, which can be used in defining templates.

2.7.2.3 inGroupId (Optional)

inGroupId is the group Id which is referenced by the *subgroupOb*. It is used when number of objects or fields in subgroup is dependent on another group of DPDL's structural part. The value of this attribute is id of another subgroup. The *inGroupId* always present when *forEach* or *inEach* attribute is used.

2.7.2.4 forEach (Optional)

The value of this attribute can be class, object, operation, function. To understand the use of *forEach,* we need to take an example of a design pattern template, in which there is a class (e.g. *class y*) which has an object of all the classes of some other subgroup (e.g. *subGroup x*). As discussed earlier, the template should handle all variation of design pattern, so different variation of the same design pattern can have different number of objects in the *class y* depending upon the number of Classes in *subGroup x.* So in template we have to say that for each class in *subGroup x,* there should be an object of it in class y. The attribute *forEach* identifies for which structural part this object is depended on. In our case it is class subgroup on which it is dependent. Now the *inGroupId* will identify the id of the group (*subGroup x*) whose number of classes it will be based on.

When *forEach* is used in the *subGroupOp* then only the *objecClass* id is changed. The numbers of objects created are equal to the number of classes in the subgroup identified in *inGroupId* and all these objects have same containing class.

2.7.2.5 inEach (Optional)

This variable is also used in conjunction with the *groupId* attribute. Whenever *inEach* is used in any type of subgroup, then there should be *inGroupId* attribute present.

inEach attribute is added to handle the situation when user wants to describe that a particular object or field is present in all the classes in a subgroup (*subGroup o*), and the value *noOfClasses* of that *subGroup o* is more than 1. *subGroup o* is the child of the classes element. There can be two cases, one scenario is that we have some numeric value (e.g. 2), in *noOfClasses* of *subGroup o.* In this case we can either show two objects, one for each class, in the DPDL, or we can show it in the DPDL by just showing one object and have the value of *inEach* attribute as class and give the id of *subGroup o* on which it is dependent on, in *inGroupId*. This way it tells the programmer that exact function which is defined in *subGroupOp* is present in each class of a particular subgroup whose value is given in *inGroupId*. The value of the attribute *inEach* is class as it is referring to a subgroup which is child of classes element.

Second case is when we are defining a design pattern and the value of *noOfClasses* of *subGroup o*, is "*one-to-many*". In this case we don't know the exact number of classes. So we require a way to mention that this object is dependent on a particular subGroup which is *subGroup o,* and each class of *subGroup o* has the object described in this sub group.

2.8 Relationship Element

One last piece of important information for any class diagram structure and especially for the design pattern structure is the relationship between classes. The relationship tells how the classes are going to interact with each other. Many design patterns differ only on the basis of relationship between the classes.

Relationship element also contains SubgroupR element whose child is Relation which contains relationship information between two classes. The SubgroupR element is

extending the capability of Relation element to handle templates therefore the SubgroupR element and its attributes are described after the Relation Element.

2.9 Relation Element

The relation element is the element in the schema in which each individual unique relationship between two classes is described. The attributes of the relation element are related to describing each relationship accurately and completely. They are kept simple and easy to describe. Below in Figure 8 the relation element is shown graphically.

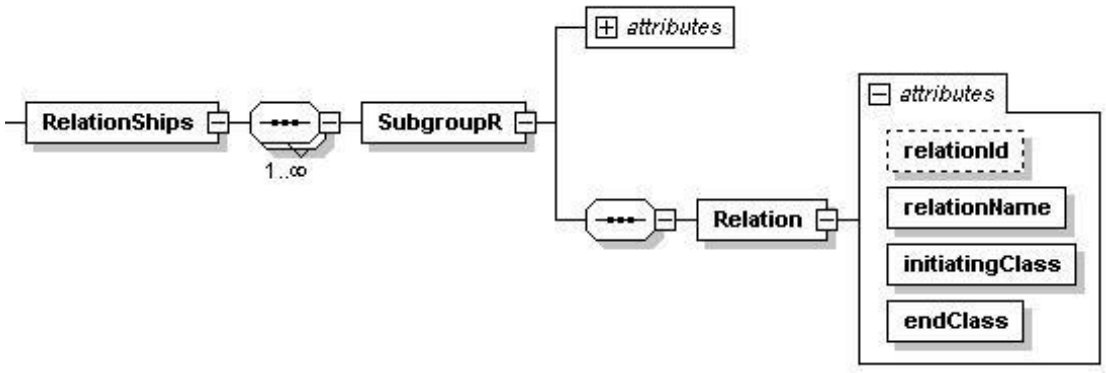

Figure 8: Attributes of Relation Element in DPDL

2.9.1 Relation Element Attributes

2.9.1.1 relationId (Optional)

The attribute relationId is for identifying the relation between two classes uniquely. Identification of the relation is the sole purpose of it.

2.9.1.2 RelationName (Mandatory)

The relationName attributes identifies the name of the relationship. This attribute also have predefined values from Association, Generalization, Aggregation, Composition, Dependency, Realization and Nesting. In future more relationship types can also be added.

2.9.1.3 initiatingClass (Mandatory)

Each relation is between two classes exactly. The initiating class is the class starting the relationship or is invoking a relation.

2.9.1.4 endClass (Mandatory)

The class which is invoked is identified in endClass. The id of the class which is on the receiving end is given in endClass attribute.

2.9.2 SubGroupR Element Attributes

The major use of subgroupR element is also in the template. Therefore its attributes are optional as the instance of design pattern can also be created in DPDL without using these attributes.

2.9.2.1 groupId (Optional)

The unique id of any group is mentioned in this attribute. With this unique group id this group can be referenced in any part of the DPDL and dependency between one part to another part can be created.

2.9.2.2 noOfRelations (Optional)

This attribute define how many identical relations are in the design pattern like the ones mentioned in this group. This attribute can have numeric as well as textual value. So we can have values like 1, 3 or 5 etc, also a value like "*one-to-many*" is acceptable, which can be used in defining templates.

2.9.2.3 inGroupId (Optional)

inGroupId is the group Id which is referenced by the *subgroupR*. It is used when the number of relationship in subgroup is dependent on another group of DPDL's structural part, then the value (id of another subgroup) in this attribute will be used to identify the independent group. The *inGroupId* always come when *forEach* is used.

2.9.2.4 forEach (Optional)

The value of this attribute can be class, object, operation, function, but in subGroupR its almost always class. To understand the use of *forEach*, we need to take an example of a design pattern template, in which there is a parent class which can have many child classes. The child classes belong to different subgroup (e.g. *subgroup R*). As discussed earlier, the template should handle all variations of design pattern, so different variations of the same design pattern can have different number of child classes in *subgroup R*. So in template we have to show that the relation between parent and all child classes is a generalization relationship. The *inGroupId* will identify the id of the child classes subgroup which is *subgroup R* in our case. The numbers of relationships in the realization of the pattern is equal to the number of child classes based on *subgroup R's noOfClasses* value.

2.9.2.5 changingClass

This attribute should always be used with *forEach* attribute. It is used when with one relationship information we want to give information about large number of identical relationship. So we have a value in *inGroupId*, identifying which class is changing, but we also need to identify which end this class belongs in the relationship i.e. is it initiating class or the end class. So this attribute has the value of either initiating class or the end class..

3. BEHAVIORAL ATTRIBUTES

Behavioral Attributes of a design pattern are contained in the behavioralAttribute element. Behavioral attribute covers how the classes are interacting and how they invoke each other and achieve the desired objective of the design pattern.

As the first target, the unique behavioral function of design patterns were identified, then we tried to create a recursive solution in XML which can allow any combination of behavioral aspect to be described in DPDL.

Another important aspect which we have to remember in the behavioral attributes is that the sequence is very important. For structural attributes of a design pattern, the sequence of writing different objects or functions or relationships do not matter as in the end the result will be the same. But in behavioral elements the sequence is important to inform the correct behavior.

3.1 Overview of BehavioralAttributes Element

BehavioralAttribute element in our schema contains five elements. These elements are SetObject, call, create, loop and condition. Each of the elements can call the other behavioral aspect inside it. This gives the flexibility to have any sort of combination to describe the design pattern. It is also a good flexibility for specifying future design patterns as it does not pose any limitation on the design patterns.

BehavioralAttribute is just a big container which is keeping all the behavioral elements in it. The behavioralAttribute element in itself does not have any attribute; it has other for other elements which are describing the behavioral aspect of the design pattern.

Graphically the structure is represented in Figure 9

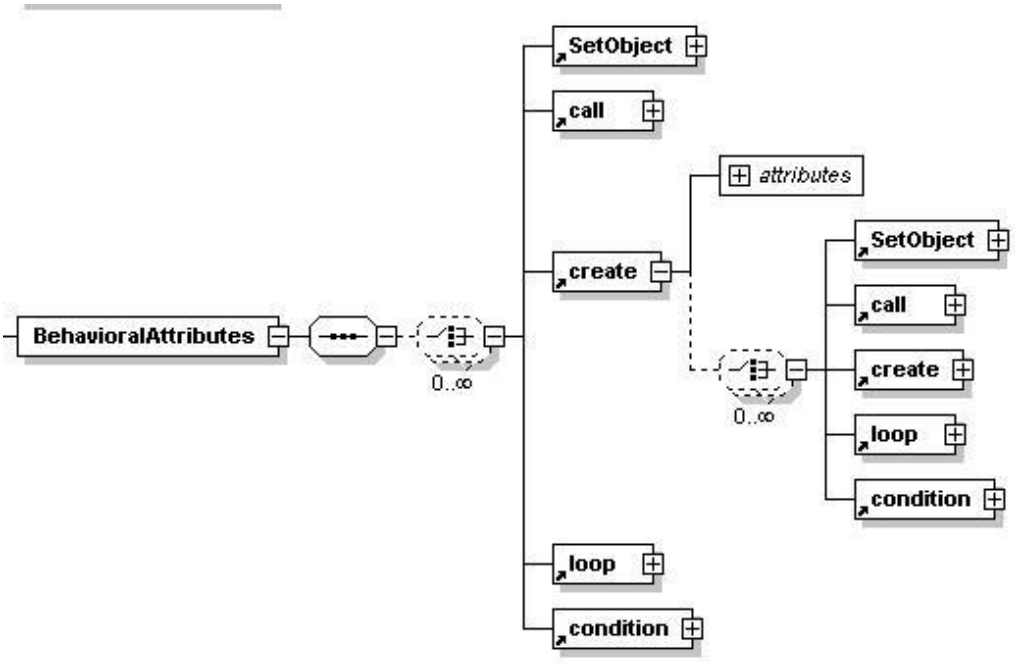

Figure 9: DPDL's Behavioral Attributes

There are three special common attributes in each BehavioralAttribute Elements for handling the templates. These attributes are forEach, inEach and inGroupId. These attributes are optional and will not be discussed in each Element of the BehavioralAttributes separately.

3.2 SetObject Element

SetOject attribute is for assigning a variable or object with some other object. In developer's term it represents typecasting of one object into another object or object type. Typecasting is quite commonly used in different design patterns.

3.2.1 SetObject Element's Attributes

Following are the attributes of SetObject Element. The attributes are shown in schematic diagram in Figure 10

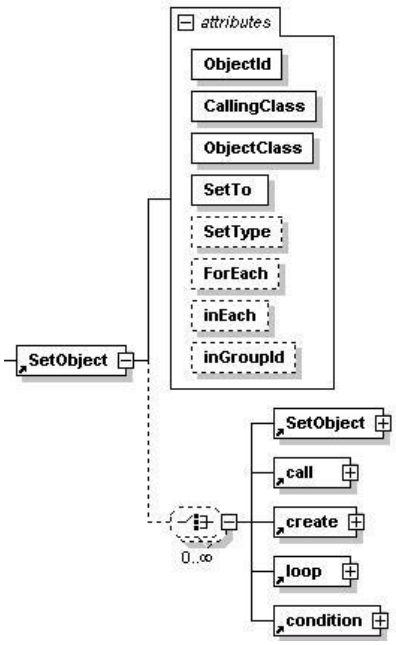

Figure 10: SetObect Element's Attributes in DPDL

3.2.1.1 CallingClass (Mandatory)

The class that contains this behavior of typecasting of object to some other object is identified in the callingClass attribute.

3.2.1.2 ObjectClass (Mandatory)

The attribute ObjectClass describe the current class of the object to which it belongs.

3.2.1.3 ObjectId (Mandatory)

The attribute ObjectId is the unique identifier of the object.

3.2.1.4 SetTo (Mandatory)

The SetTo attribute identifies the new Type to which the object is set.

3.2.1.5 SetType (Optional)

The SetType attribute identifies if the object is being changed through an object or through a class. So its value can either be object or class.

3.3 Call Element

Call is the most widely used behavioral element. Whenever a function is invoked in a design pattern, call element is used to capture it. The attributes of Call elements are as follow:

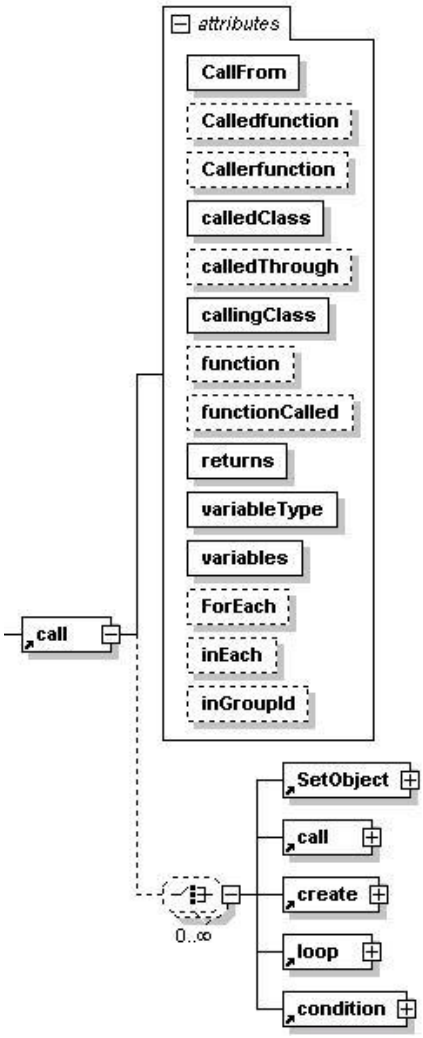

Figure 11: Call Element's Attributes in DPDL

3.3.1 Call Element's Attributes

3.3.1.1 CallFrom (Mandatory)

The CallFrom attribute identifies from which class the call is invoked. So the calling class id is given as the value of CallFrom

3.3.1.2 CalledFunction (Optional)

The CalledFunction is the attribute which stores the name of the function which is being called.

3.3.1.3 CallerFunction (Optional)

If the call to the function is made from inside another function, then the name of the function from which the CalledFunction is invoked is stored in CallerFunction attribute

3.3.1.4 CalledClass (Mandatory)

The CalledClass attribute identifies the class of the CalledFunctions. The value of the called class is saved in the CalledClass attribute.

3.3.1.5 CalledThrough (Optional)

Each function can be called through either directly or through some object. If the function is not called through any object then the value of the CalledThrough is null, otherwise the name of the object is given, through which the function is called, in this attribute.

3.3.1.6 CallingClass (Mandatory)

The class from which the function is called is identified in this attribute.

3.3.1.7 VariablesPassed (Mandatory)

Some functions require some input variables also. The variable name for these functions is passed through this VariablePassed attribute.

3.3.1.8 VariabledTypes (Mandatory)

If the invoked function has input variables then the type of those variables is identified in this attribute.

3.3.1.9 Returns (Mandatory)

The returns attribute identifies the object type which is returned by the invoked function.

3.4 Create Element

The create element is the third behavioral element. This element is used for depicting the creation of some object in the design pattern. The attribute of create element identifies the creation properties.

3.4.1 Create Element's Attribute

The attribute in the create elements are listed below with the description. The graphical schematic representation is shown in the Figure 12

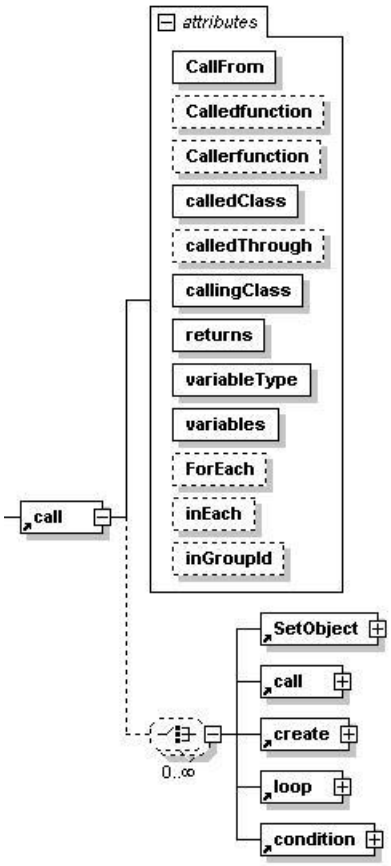

Figure 12: Create Element's Attributes in DPDL

3.4.1.1 ObjectId (Mandatory)

The attribute objectId identifies the object. Whenever a new object or variable is declared it is given a unique identifier.

3.4.1.2 createType (Mandatory)

This attribute identifies if the createType is new.

3.4.1.3 Collection (Mandatory)

If the object which is being created is some sort of array then the collection attribute will have the value as Yes otherwise it has the value as No.

3.4.1.4 CallingClass (Mandatory)

The class in which the object is created is identified in CallingClass attribute of the Create Element.

3.4.1.5 ObjectClass (Mandatory)

The class of the object which is being created is identified in the ObjectClass attribute.

3.4.1.6 Returns (Mandatory)

Sometime the object returned by called class is not the object of the called class but it is an object of the some other class. In that case the returns will be different then objectClass

3.4.1.7 Variables (Optional)

For creating an object, sometime variables are also required to be passed. The name of these variables will be given in the Variables attribute.

3.4.1.8 variableTypes (Optional)

The type of the variables which are passed is given in the variableTypes

3.5 Loop Element

Loop is another important behavioral element for the DPDL. All the loop which are the part of the design pattern are described through loop element. The loop element has following attributes

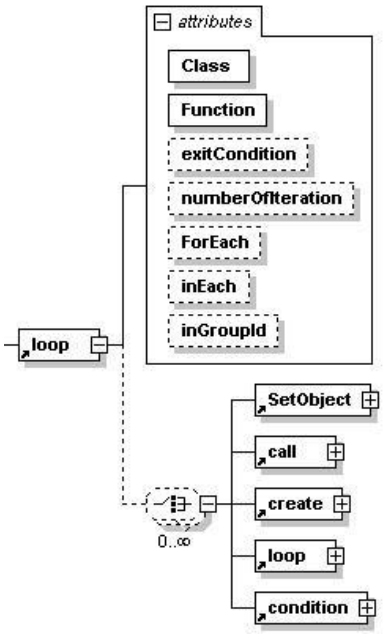

Figure 13: Loop Element's Attributes in DPDL

3.5.1 Loop Element's Attributes

3.5.1.1 Class (Mandatory)

This attribute contains the name of the class in which this loop is present.

3.5.1.2 Function (Mandatory)

If the loop is inside the function then the name of the function is given in the Function.

3.5.1.3 ExitCondition (Optional)

This attribute contains the condition on which the loop will terminate.

3.5.1.4 numberOfIterations (Optional)

If the number of iteration is fixed then this attribute can have a numerical value.

3.6 Condition Element

Behavior of design pattern is not always sequential. In those cases on the basis of some condition the sequence of the program is interrupted which is represented by conditional statement in programming language. For this we have condition element in our DPDL

3.6.1 Condition Element's Attributes

The attributes for condition elements are shown in the below Figure 14

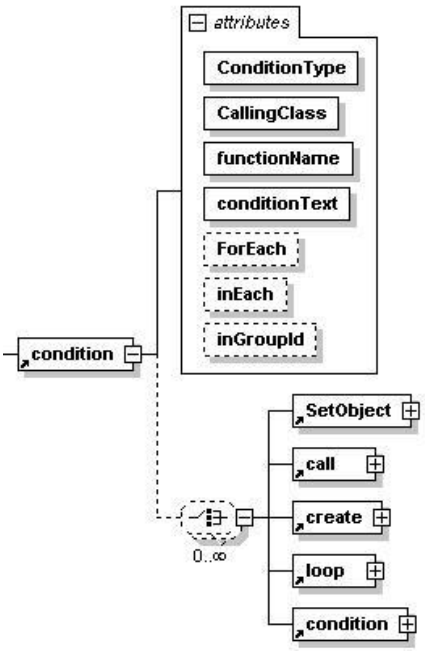

Figure 14: Condition Element's Attributes in DPDL

3.6.1.1 ConditionType (Mandatory)

The condition structure in most cases has two branches. Sometime they are called as normal code path and alternate code path. The conditionType tells which of the code path this condition is representing. The conditionType can be extended to more than 2 options. It should also be mentioned that the correct representation of ConditionType is in the hands of the end user. For example if a alternate code path is represented without first mentioning the normal code path then the language DPDL will consider it correct, where as in normal environment this will be considered as a bug. The reason for not handling is that XML does not provide very fine grained control to handle such a condition. Moreover the benefit is that this allows easy extendibility for handling more than two code paths. Switch condition can also be supported by ConditionType.

3.6.1.2 CallingClass (Mandatory)

The name of the class in which this condition is present is mentioned in the callingClass attribute

3.6.1.3 FunctionName (Mandatory)

The name of the function in which the condition is used is mentioned in the functionName attribute

3.6.1.4 conditionText (Mandatory)

The statement or text of the condition is mentioned in conditionText attribute.

3.7 Common Attributes

3.7.1 inGroupId (Optional)

inGroupId is the group Id of another ***structural*** part of DPDL, which is referenced by any behavioral element. It is used when an action (behavioral element like call or created) is dependent on another structural part, then this attribute is used to identify the independent group. The *inGroupId* always come when forEach or inEach attribute is used. Its use with *forEach* n *inEach* is explained in Section 3.7.2 and Section 3.7.3 respectively.

3.7.2 forEach (Optional)

The value of this attribute can be class, object, operation, function. To understand the use of *forEach,* take an example of a design pattern template in which there is a create call for all the classes in a particular subgroup (e.g. *subGroup x*). As discussed earlier, the template should handle all variation of design pattern, so different variation of the same design pattern can have different number of classes. So in template we have to say that for each class in *subGroup x*(where *x* is id group of independent subGroup), there should be a create call in classA. *forEach* attribute identifies for which structural part this create call is repeated for. In our case it is class for which it is repeated for. Now the *inGroupId* identifies the id of the group (which is *subGroup x*) whose number of classes will determine the number of create calls.

3.7.3 inEach (Optional)

This variable is also used in conjunction with the *groupId* attribute. Whenever *inEach* is used in any type of subgroup, then there should be an *inGroupId* attribute present.
inEach attribute is added to handle the situation when user wants to describe that a particular behavioral action (e.g. call) is present in all the classes of a subgroup (e.g. *subGroup x*). The value *noOfClasses* of *subGroup x* should be more than 1. In this case we can show it in the DPDL by just showing one call element with all the attributes and have the value of *inEach* attribute as class and give the id of *subgroup x* to *inGroupId* in the call element. This way we are showing that exact call is present in each class of a *subGroup x.* The value of the attribute *inEach* is class as it is referring to a subgroup which is child of classes element.