CHAPTER VII

UML SEQUENCE DIAGRAM & TEST CASE GENERATION

7.1 Overview

In chapter 5, patterns of polymorphic assignment are discussed. Although how to recognize a pattern is explained, it does not cover how to generate test cases for a particular interaction. To generate test cases, more detailed information is required. This chapter picks up where chapter 5 leaves off by explaining about how a UML sequence diagram is extended to support test case generation and also about how test cases are generated from the extended sequence diagram.

7.2 Extension to UML Sequence Diagram

It is necessary that UML sequence diagram is extended to support our approach. From chapter 5, it is possible to identify polymorphic assignment pattern from a given interaction. However, the pattern alone is not enough for test case generation. In order to generate test cases, a condition which an object of a particular subclass is assigned in a polymorphic assignment must be known. It is obvious that a UML sequence diagram alone does not provide this kind of information. An extension to UML sequence diagram is required to support this information.

In this research, tagged values are used for extension mechanism. Tagged value is a standard element in UML semantic. A tagged value element can be attached to any UML element to describe a property which is not supported by standard UML semantic. It comes in a form of a key (tag name) and a value; therefore, any kind of property can be expressed. In this research, a set of tagged value elements are proposed. These tagged value elements are for representing the information necessary for test case generation of polymorphic interaction. A polymorphic interaction with proper usage of these tagged value elements is sufficient for test case generation, regarding the polymorphism issue.

Table 7.1 summarizes all tagged values proposed by the research. Note that not all of them are necessary for a particular situation. For a certain pattern of polymorphic assignment, a particular set of the tagged values are required. Subsequent subsections discuss about usage of the tagged values for each polymorphic assignment pattern.

Table 7.1 Tagged values proposed in the research

Name	Attached to	Description
assigned-to	an Argument in a CallAction	This tagged value is for identifying that a value/object of an argument is assigned to a role/variable whose name is the same as the value of this tag. This tagged value is only applicable to an Argument of a CallAction at the beginning of an interaction (from Actor).
factory-method	an Operation	This tagged value specifies that an operation is a factory method which returns an object from a particular class hierarchy. The value identifies the root superclass of a hierarchy. The value is not necessarily important, since the return type is already specified by the operation.

Table 7.1 Tagged values proposed in the research (cont.)

Name	Attached to	Description
factory-method-param	a Parameter in an operation	This tagged value identifies that a parameter affects the behavior of the factory-method operation, of which it is a part. The value is unnecessary if the parameter is defined in a factory-method operation. There is an exceptional case where the tag is defined for a parameter which does not belong to a factory-method operation. In this case, the parameter affects another factory-method operation on the same class. Therefore, the value of the tag is required to identify on which factory-method operation it has an effect.
derived-from	an Argument in a CallAction	This tagged value specifies from which variable this argument derives a value from. The value specifies the name of the variable.

Table 7.1 Tagged values proposed in the research (cont.)

Name	Attached to	Description
return-to	a CallAction	This tagged value is for identifying that the value/object returned from the CallAction element is assigned to a role/variable whose name is the same as the value of this tag.
factory-method-condition	an Operation	This tagged value specifies condition of a factory method.
environment-variable	a Collaboration	This tagged value identifies environment variables which affect the collaboration.
scenario-condition	a Collaboration	This tagged value specifies a predefined condition of the scenario represented by the interaction.

From the table, a name of a tagged value element is presented along with the name of UML element it can be attached to. This is very important, since a certain piece of information must be attached to a corresponding element in order to make sense. The description column describes the purpose of each tagged value. The detail of each tagged value is discussed along with its usage in the following subsections. The example from chapter 5 is repeated here to show how the extension is applied for each polymorphic interaction pattern. The example uses the inheritance hierarchy shown in figure 7.1.

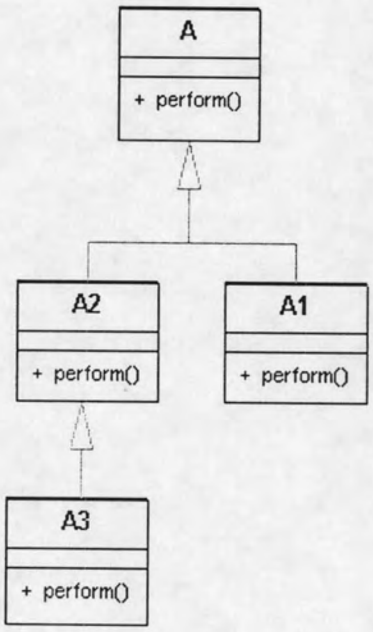

Figure 7.1 Example of a UML class diagram

7.2.1 Simple Polymorphic Assignment

Simple polymorphic assignment pattern is rather straightforward. An object is assigned from an input of a polymorphic interaction to a polymorphic entity. There is only one tagged value required for this pattern of polymorphic assignment: assigned-to. This tagged value identifies which input argument is assigned to the polymorphic entity.

As stated in its description, "assigned-to" tagged value is only applicable to an Argument element of the CallAction element from an actor. In other words, it can only be attached to an argument of the initiator message of an interaction. The usage of the extension for Simple polymorphic assignment is illustrated in figure 7.2.

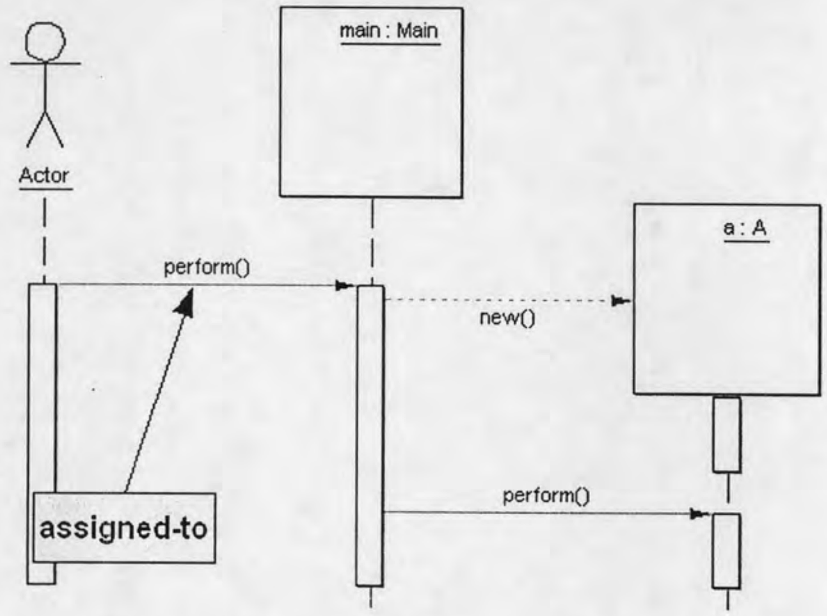

Figure 7.2 Usage of extension for Simple polymorphic assignment

7.2.2 Parameter-Influenced Polymorphic Assignment

Parameter-influenced polymorphic assignment pattern is not as straightforward as Simple polymorphic assignment pattern. It relies on a factory method which returns an object to be assigned to the polymorphic entity, and the factory method relies on a set of parameters which are supplied as interaction arguments.

The polymorphic assignment pattern requires a set of tagged values to express its behavior. First of all, an operation must be attached with "factory-method" tagged value to identify that it is a factory method for a particular class hierarchy. An operation is not essentially an element in an interaction, since it belongs to a class which is a structural element. However, an operation marked as a factory method is always a factory method for any interaction. Therefore, placing the tagged value at an operation is sufficient.

In addition to "factory-method" tagged value, "factory-method-condition" tagged value must be attached to the operation. The value indicates the condition which an object of each subclass is returned from the method. The format of condition along with an example is shown in figure 7.3.

```
<class-name>:<condition>;<class-name>:<condition>:…

Example
A: (x>0) and (x<100);B: (x>=100) and (x<200);C: (x>=200) and (x<300)
```

Figure 7.3 Format of condition

A value of "factory-method-condition" comprises of one or, usually, more conditions of a class to be returned. If there is more than one condition, they are separated by a semicolon. For each condition, there are a name of a class of an object to be returned and a condition clause. The format of condition clause follows OCL (Object Constraint Language) grammar [7]. The example in figure 7.1 shows that an object of class A is returned if the value of 'x' is greater than 0 and less than 100, or an object of class B is returned if the value of 'x' is greater than or equal to 100 and less than 200, or an object of class C is returned if the value of 'x' is greater than or equal to 200 and less than 300. Note that since the condition format is in OCL, any kind of condition is practically possible.

Variables used in conditions in a "factory-method-condition" tagged value for parameter-influenced polymorphic assignment are parameters of the factory method. It is required that these arguments are identified. A "factory-method-param" tagged value is for this purpose. It is attached to a parameter which is a part of conditions of the factory method. It is not necessary that the parameter is a parameter of the factory method itself. It is possible that the parameter is a parameter of another operation on the same class, but it is a factor in the conditions of the factory method. See the interaction in figure 5.6 as an example.

To know to which role an object returned from a factory method is assigned, a "return-to" tagged value is necessary. It is attached to a CallAction element, which is a call to a factory method. The value identifies the name of role to which the returned object is assigned.

Similar to Simple polymorphic assignment, an "assign-to" tagged value is attached to the CallAction element which is an initiator message of an interaction. However, here it is not for a direct assignment of a polymorphic entity. It is for an assignment of an argument of a call to a factory method. Since it is possible that there is more than one factory method in an interaction, and they may declare parameters with the same name, direct assignment from interaction inputs to arguments may be ambiguous. For example, a factory method may be called twice in an interaction with different arguments, which results in 2 polymorphic assignments. To solve this issue, an "assign-to" tagged value is an assignment to a variable which is global in an interaction. Each argument of a CallAction element, which is a call to a factory method, must be attached with a "derived-from" tagged value in order to be assigned a value to be used in the conditions of the factory method. The value of "derived-from" is a variable name which the argument derives a value from.

Figure 7.4 shows the usages of the tagged value extension for Parameter-influenced polymorphic assignment. Note that the boxes with no arrow are not directly associated with an element in a UML sequence diagram.

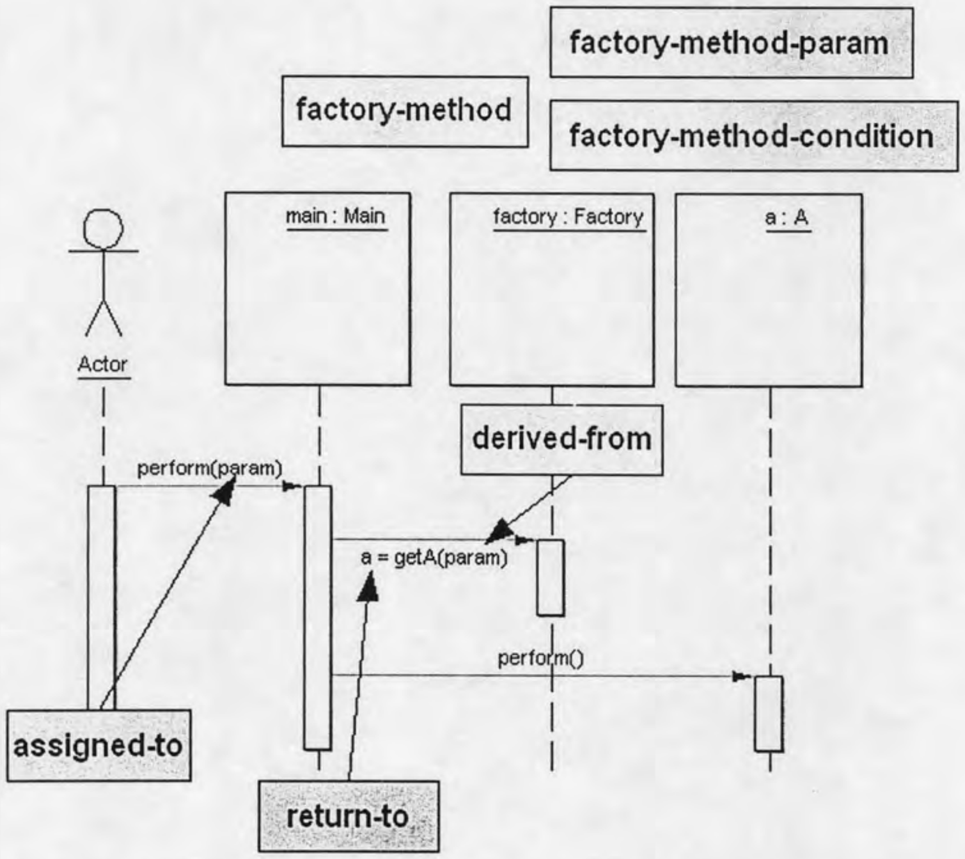

Figure 7.4 Usage of extension for Parameter-influenced polymorphic assignment

7.2.3 Configuration-Influenced Polymorphic Assignment

In some ways, Configuration-influenced polymorphic assignment pattern is similar to Parameter-influenced polymorphic assignment pattern. They both rely on factory methods. The major difference is a factory method in Configuration-influenced polymorphic assignment does not take an argument. Its behavior is complete influenced by factors which are not controlled by an interaction.

With the similarity, the pattern shares some tagged values with Parameter-influenced polymorphic assignment pattern. The "factory-method", "factory-method-condition", and "return-to" tagged values are applied in the pattern as in parameter-influenced polymorphic assignment pattern. However, the conditions do not contain factors which are arguments. These factors are declared in an "environment-variable" tagged value, which is attached to a Collaboration element. A Collaboration element is the root element of an interaction in UML semantic. The format of environment variable declaration is shown in figure 7.5.

```
<type> <name>;<type> <name>;…

Example
int x ;java.lang.String y ;java.lang.Object z
```

Figure 7.5 Format of environment variable declaration

Declarations have a semicolon as a delimiter. Each declaration consists of a type and a variable name. From the example, there are 3 environment variables: 'x' as an integer, 'y' as a String in Java, and 'z' as an object of any type.

The usage of the tagged value extension for Configuration-influenced polymorphic assignment is shown in figure 7.6.

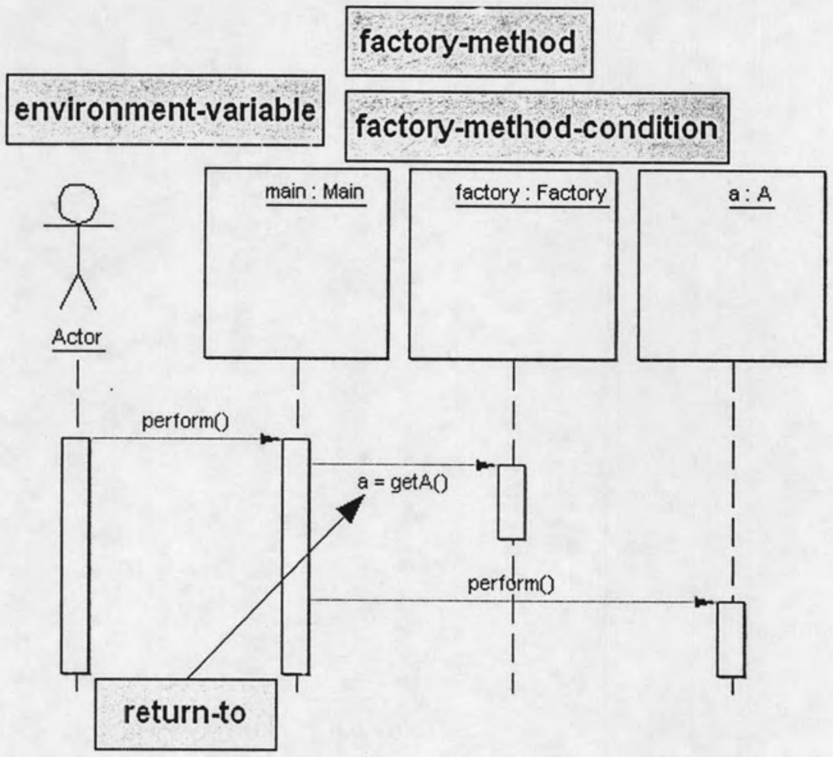

Figure 7.6 Usage of the extension for Configuration-influenced polymorphic assignment

Table 7.2 summarizes usage of tagged values for each pattern of polymorphic assignment.

Table 7.2 Tagged values used for each polymorphic assignment pattern

Pattern	Tagged Value(s)
Simple Polymorphic Assignment	assigned-to
Parameter-Influenced Polymorphic Assignment	assigned-to factory-method factory-method-param derived-from return-to factory-method-condition
Configuration-Influenced Polymorphic Assignment	factory-method return-to factory-method-condition environment-variable

7.3 Test Case Generation

From the extension to UML in the previous section, it is possible to generate test cases from UML sequence diagrams with proper use of the extension.

It is assumed that a particular adequacy criterion is selected from the adequacy criteria in chapter 4. Therefore, it is known which classes are to be tested for a certain test scenario before a test case for the test scenario is generated.

Test case generation usually involves a great deal of test data generation techniques which are out of the scope of the research. The test case generation discussed here is neutral to test data generation. It is assumed that test data can be generated from any given condition. A certain data generation technique may be employed, or the data may be randomly generated and evaluated against the condition.

7.3.1 Simple Polymorphic Assignment

Test case generation for Simple polymorphic assignment is based on "assigned-to" tagged values. After a polymorphic entity is identified, an "assigned-to" tagged value which is an assignment to the polymorphic entity must be located. The argument, to which the "assigned-to" tagged value is attached, is the focus of test case generation. Objects of different subclasses must be used as test inputs according to polymorphic adequacy criteria. .

Although test case generation for the pattern is simple, it is not complete. It is impossible to identify how an object as a test input is created from the interaction diagram. An object of a particular class may have a certain way to be created. Moreover, an object, which is used as a test input for a particular test scenario, may also require a certain way to be created. This is beyond the scope of the research.

The example in chapter 5 is repeated here to show how test cases can be generated for simple polymorphic assignment pattern. Figure 7.7 shows a UML class diagram which illustrates the inheritance hierarchy of abstract class Product, and figure 7.8 shows a UML sequence diagram of product purchasing process, which contains Simple polymorphic assignment.

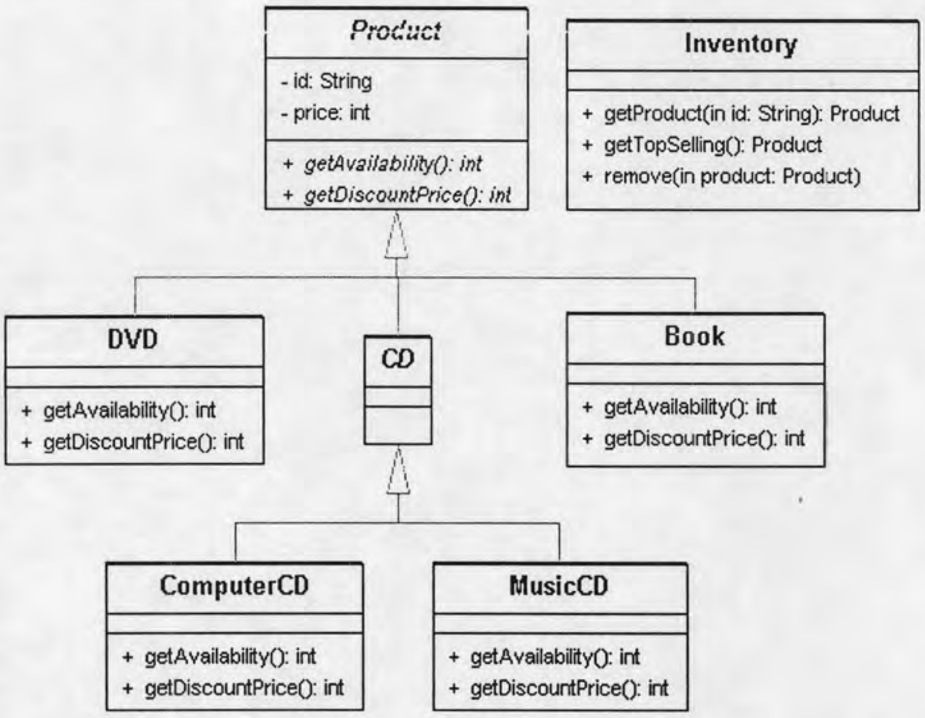
Product
- id: String
- price: int
+ getAvailability(): int
+ getDiscountPrice(): int
Inventory
+ getProduct(in id: String): Product
+ getTopSelling(): Product
+ remove(in product: Product)
DVD
+ getAvailability(): int
+ getDiscountPrice(): int
CD
Book
+ getAvailability(): int
+ getDiscountPrice(): int
ComputerCD
+ getAvailability(): int
+ getDiscountPrice(): int
MusicCD
+ getAvailability(): int
+ getDiscountPrice(): int

Figure 7.7 UML class diagram for store example

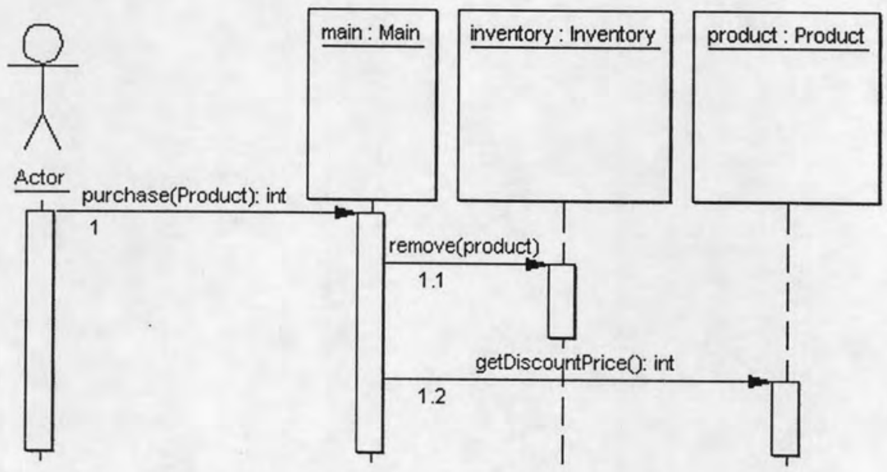

Figure 7.8 UML sequence diagram for purchasing product

From the UML sequence diagram, the argument 'product' of the CallAction of the first message from the actor to object 'main' is attached with a "assigned-to" tagged value. The tagged value has a value as "product". This shows that the argument is assigned to role 'product', which is a polymorphic entity.

Test cases generated for the interaction is shown in table 7.3. It is very similar to the result from the example in chapter 5. As stated earlier, it is impossible to identify how an input object is created from the currently available information.

Table 7.3 Test cases for the interaction in figure 7.8

No.	Test Input (product argument)
1	An instance of Book
2	An instance of ComputerCD
3	An instance of MusicCD
4	An instance of DVD

7.3.2 Parameter-Influenced Polymorphic Assignment

Test cases for an interaction with Parameter-influenced polymorphic assignment pattern are generated based on the condition of the factory method, represented by "factory-method-condition", which returns to the polymorphic entity. First all factory methods in an interaction are identified with "factory-method" tagged values. Then the factory method for a particular polymorphic entity is located. The value of "return-to" tagged value helps in identifying which factory method returns to the polymorphic entity. After that "factory-method-param" tagged values are used for identified which parameters of the factory method are a part of the condition.

At this state, the condition for test case generation is known. Then a class of the returned object is selected. Then the condition of the class from the factory method condition is used for test data generation. However, the factors in the condition are not themselves inputs of the interaction. The "assigned-to" and "derived-from" tagged values are used for mapping between inputs of the interaction and the factors in the factory method condition. A "derived-from" tagged value identifies from which variable a certain argument derives a value. It is then mapped to a value from a "assigned-to" tagged value, which identifies to which variable an interaction input assigns value. These 2 types of tagged value maps test data generated for factors in a factory method condition to test data generated for interaction test inputs.

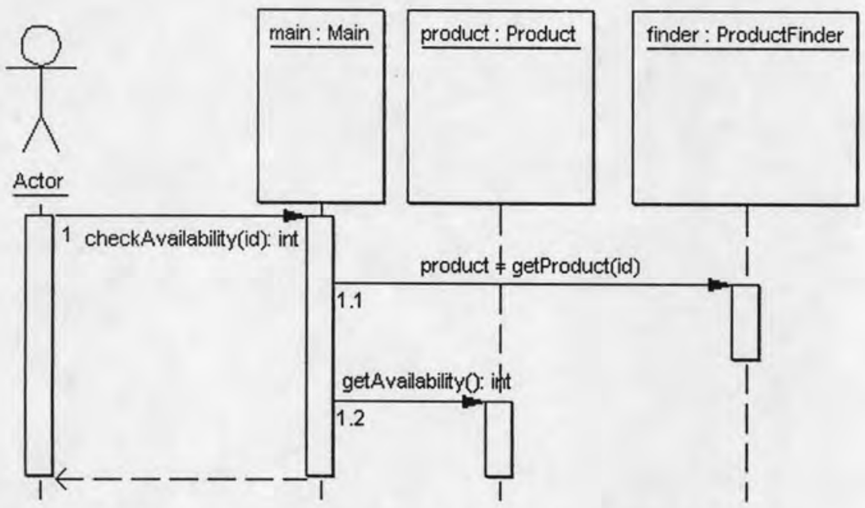

Figure 7.9 UML sequence diagram for checking product availability

Figure 7.9 repeats the example from chapter 5. The interaction in the figure contains a Parameter-influenced polymorphic assignment. The interaction input 'id' is used as an argument of the call to the method 'getProduct', which is a factory method. An "assigned-to" tagged value is attached to the argument 'id' of the CallAction element from the actor to the object 'main', and a "derived-from" tagged value is attached to the argument 'id' of the CallAction element of message 1.1 (a call to method 'getProduct'). To map to each other, both tagged values must have the same value.

The method 'getProduct' on class ProductFinder has "factory-method" and "factory-method-condition" tagged values attached. Also its parameter 'id' has a "factory-method-param" tagged value attached to show that the parameter affects the behavior of the method. The "factory-method-condition" tagged value expresses the conditions for each class of the returned object. Figure 7.10 shows the condition of the factory method. It shows that in order for the method to return an object of class Book, the argument 'id' must have its value greater than 0 and less than 10,000,000. For an object of class ComputerCD to be returned, the argument is must have its value greater than or equal to 10,000,000 and less than 20,000,000.

```
Book: (id > 0) and (id < 10000000); ComputerCD: (id >= 10000000) and
(id < 20000000); MusicCD: (id >= 20000000) and (id < 30000000); DVD:
(id >= 30000000) and (id < 40000000)
```

Figure 7.10 Factory method condition for the UML sequence diagram in figure 7.9

The CallAction element at message 1.1 in the interaction in figure 7.9 must be attached with a "return-to" tagged value to identify which role the object returned from the factory method is assigned to. In this example, the value is "product", which is the role of the only polymorphic entity in the interaction.

Test cases generated for the interaction is shown in table 7.4 below. Note that the values of test data are randomly generated to satisfy the conditions of test input for each test case.

Table 7.4 Test cases for the interaction in figure 7.9

No.	Test Input (id argument)
1	1,870,223
2	19,739,911
3	26,025,251
4	39,739,143

7.3.3 Configuration-Influenced Polymorphic Assignment

As stated earlier, Configuration-influenced polymorphic assignment pattern is very similar to Parameter-influenced polymorphic assignment pattern. The usage and recognition of "factory-method", "factory-method-condition", and "return-to" tagged values are exactly the same.

Since the pattern does not rely on interaction inputs like Parameter-influenced polymorphic assignment pattern, other tagged values used in the latter pattern are not applicable here. In other words, only "factory-method", "factory-method-condition" and "return-to" tagged values are almost sufficient for test case generation.

Test generation process is practically the same as Parameter-influenced polymorphic assignment, with an exception of how the factors in the condition are mapped. For this pattern, the factors are mapped to environment variables instead of interaction inputs. The value of an "environment-variable" tagged value attached to the Collaboration element is used for identifying data types of environment variables.

Figure 7.11 shows the UML sequence diagram shown previously in chapter 5. It contains a Configuration-influenced polymorphic assignment.

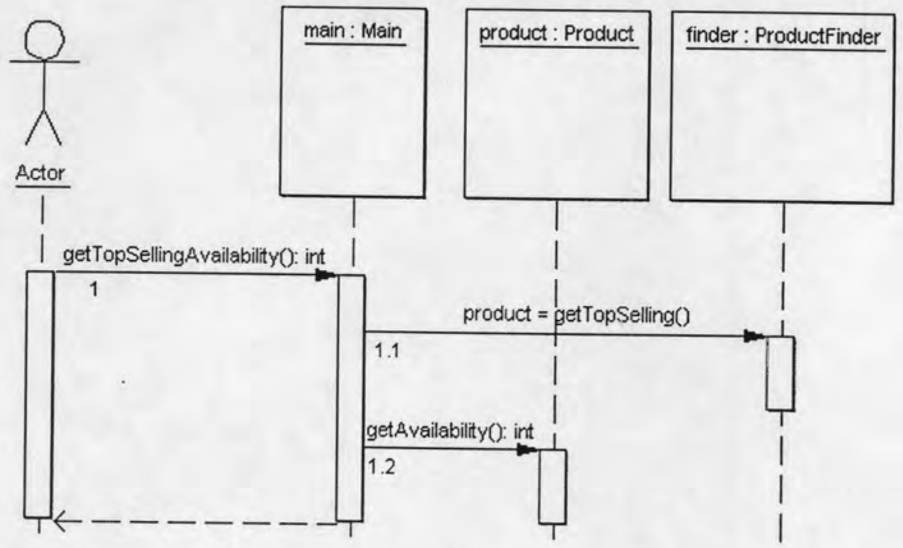

Figure 7.11 UML sequence diagram for checking availability of the top selling product

From figure 7.11, the method 'getTopSelling' is attached with a "factory-method" tagged value and a "factory-method-condition" tagged value. In addition, a "return-to" tagged value is attached to the CallAction element of the message call 1.1, which is the polymorphic assignment. The value of "return-to" tagged value is 'product', which is the role to which the returned object is assigned. The value of "factory-method-condition" tagged value is shown in figure 7.12 below.

```
Book: topProduct="Book"; ComputerCD: topProduct="ComputerCD";
MusicCD: topProduct="MusicCD"; DVD: topProduct="DVD"
```

Figure 7.12 Factory method condition for the UML sequence diagram in figure 7.11

Test cases are generated from the condition above. However, it is necessary that types of variables are known. The "environment-variable" tagged value declares names and types of environment variables. Its value is shown in figure 7.13.

```
java.lang.String topProduct
```

Figure 7.13 Environment variable declaration for the sequence diagram in figure 7.11

Using the condition from figure 7.12, test cases are generated as shown in table 7.5. There is only one environment variable 'topProduct' which affects the behavior of the factory method; therefore, to test the interaction is to execute the program under test with different environment variable values.

Table 7.5 Test cases for the interaction in figure 7.11

No.	Environment variable (topProduct)
1	Book
2	ComputerCD
3	MusicCD
4	DVD

7.4 Discussion

7.4.1 Mixed Pattern

It is possible that a single polymorphic assignment has the characteristic of both Parameter-influenced polymorphic assignment and Configuration-influenced polymorphic assignment. In other words, a polymorphic assignment may be influenced by both parameters and configuration.

This is not a problem. The extension presented earlier in this chapter supports the case. The factors used in "factory-method-condition" are not limited to either arguments or environment variables. It is possible that factors from both sources are mixed in the condition. The "factory-method-param" and "environment-variable" tagged values help in identifying whether a factor is from an argument or an environment variable. The only restriction is that an argument and an environment variable of the same name cannot be used in a factory method condition due to name conflict problem.

7.4.2 Interaction with more than one polymorphic assignment

An interaction may contain more than one polymorphic assignment. Test case generation for such an interaction is not different from an interaction with only one polymorphic assignment. As stated earlier, classes to be tested for a test scenario are known before a test case is generated. Rather than selecting just one class for a polymorphic entity, this case requires selection of a class for each polymorphic entity. However, when there is more than one polymorphic assignment in an interaction, polymorphic adequacy criteria must be considered regarding multiple fault assumption.