4. Difficulties for Constructing Reusable Components for WfMS

In section 1.1 we introduced how object-oriented programming languages support the reusability of classes and we introduced the reusability of the components of WfMSs in section 3. In this section, we will discuss the difficulties for constructing these components with reusability we defined in section 3.

4.1 Workflow Engine

As mentioned in section 3.1, the reusability of workflow engine is to be reused immediately with limited modification. To meet different requirements, the developers may change the configuration instead of modifying workflow engine. To reach this goal, we may require not only the support of object-oriented programming language but a proper design (e.g. a compositional design).

For instance, to interpret process definition is one of the functions of workflow engine. As introduced in section 0, several model languages have been developed to represent workflow process. The interpretation function should be implemented according to the model language used by workflow engine. Now, if we want to change the model language from direct graph model to XPDL, how can we meet the change without modifying workflow engine? In Figure 13, the interpretation functions are implemented within a method of `workflow_Engine_A` class. In this case, if we want to change the model language, we will need to inherit `workflow_Engine_A` class and overwrite the method that implements the interpretation function. Although the other codes and methods in `workflow_Engine_A` class are reused, the

workflow_Engine_A class cannot be reused immediately. A compositional design is shown in Figure 14. In this design, we separate the implementation of interpretation funtion from `workflow_Engine_A` class and implement it in the subclass of `Abstract_Interpreter` class. The developers may change the subclass to meet change. This design refers to Strategy Pattern [28].The `workflow_Engine_A` class invokes the method defined in `Abstract_Interpreter` class to interpret process definition. Although the actual codes are implemented in the subclass of `Abstract_Interpreter` class, the `workflow_Engine_A` class does not relate to this subclass directly. Therefore, to change the subclass does not affect the `workflow_Engine_A` class. The difference between Figure 13 and Figure 14 will be more obvious when many functions of workflow engine need to be chaged.

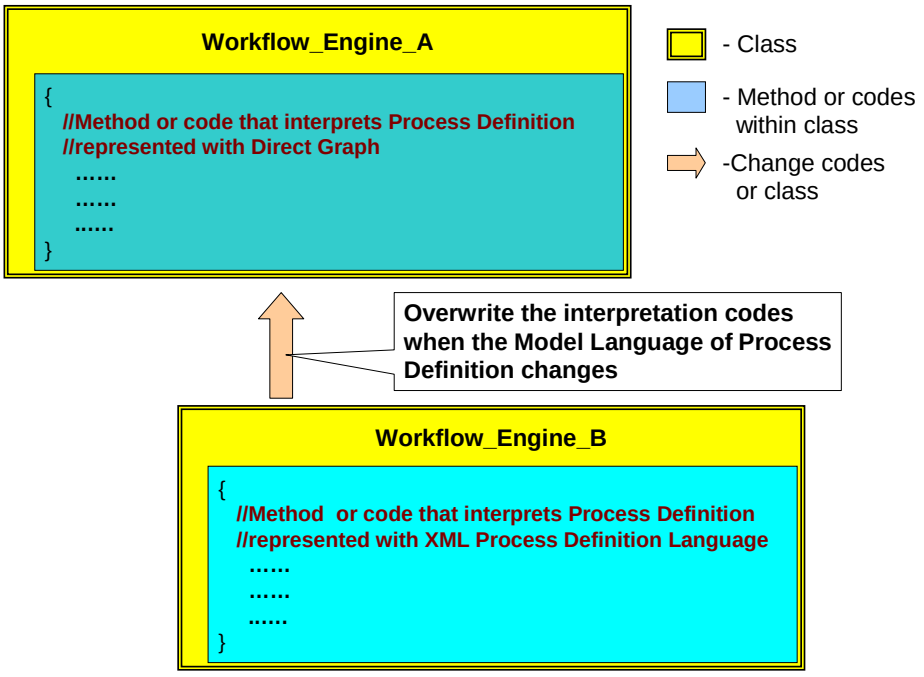

Figure 13: Change the interpretation function by overwriting codes or method.

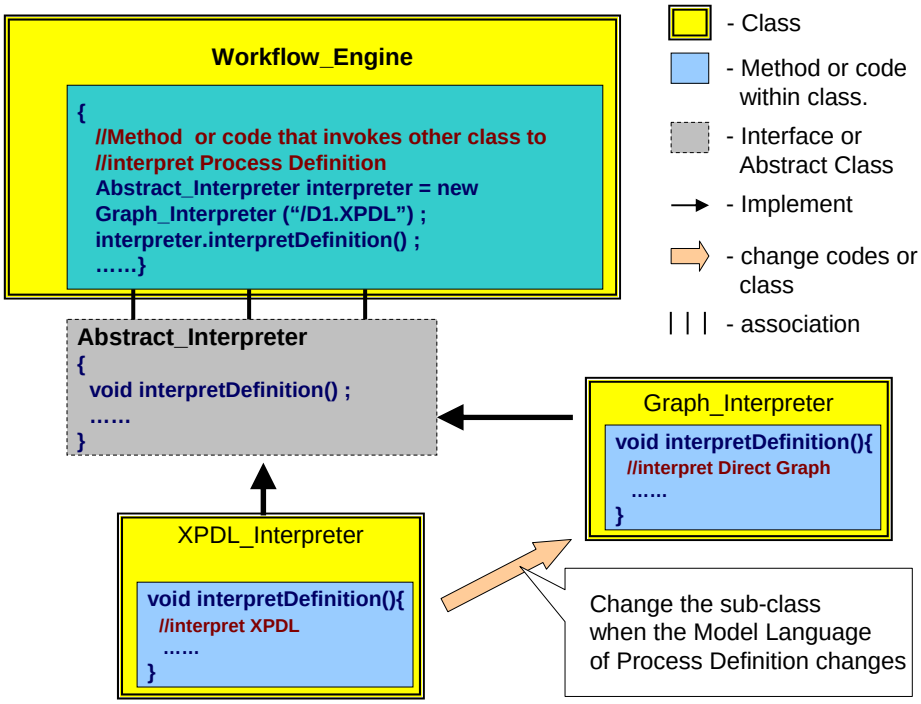

Figure 14: More flexible design for Workflow Engine.

4.2 Workflow Process

As defined in section 3.2, the reusability of workflow process is to be reused with slight or no modification and the activities belong to this process should be loaded automatically according to process definition or other configuration files before being executed. Similar to workflow engine, a compositional design may let process component more reusable.

To load and create activity instances during the execution of process automatically may need supports provided by programming languages. For example, the Java programmers can use the `forName` method in `Class` class to load specific class and use newInstance method to create instance. In addition to the support of programming language, there are some problems for us to think about. For instance, the system needs some information to load and create activity instances, but where can we get the information? And what is the proper timing for system to load activities? To deal with

these problems, we may add some advanced designs into system architecture.

4.3 Workflow Activity

The reusability of activity component is that an activity component can be reused to execute different execution code according to process definition. We can reach this goal by using object-oriented programming language, but an appropriate design (e.g. a compositional design) may be required. The design in Figure 15 is to implement the execution code in `Activity_1` class and different tasks are implemented in different `activity` class. In this case, neither activity nor execution code can be reused.

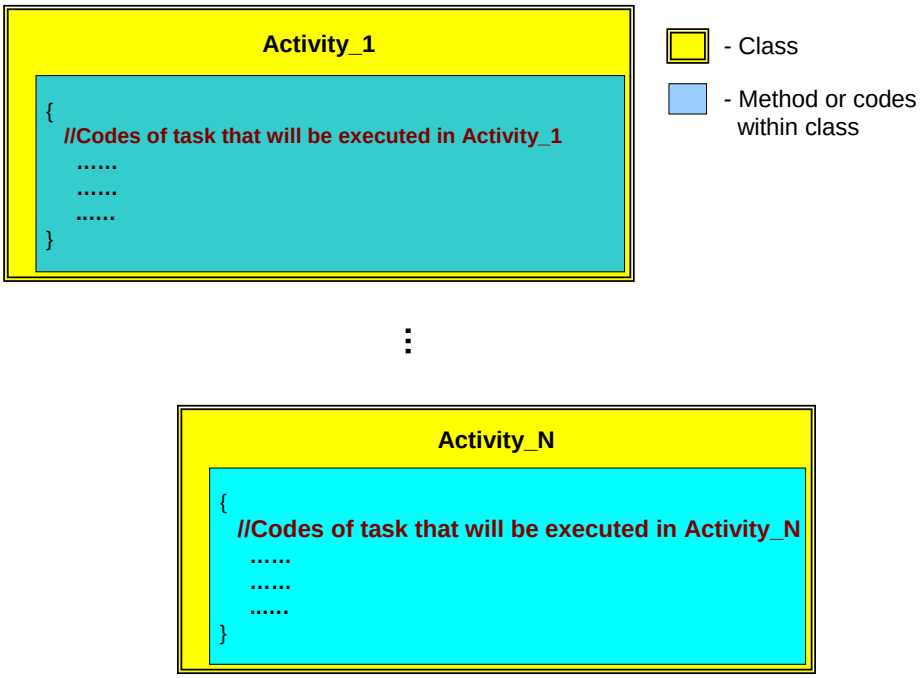

Figure 15: Change the codes of "task" by overwriting codes or method.

In Figure 16, we refer to the Strategy Pattern [28] and separate execution code from activity. A task is implemented in a subclass of ExecCode class and the subclass does not relate to `Activity` class directly. Therefore, the change of the subclass of ExecCode will not affect the `Activity` class. Also, the subclass of `ExecCode` can

be reused by other activities in different process.

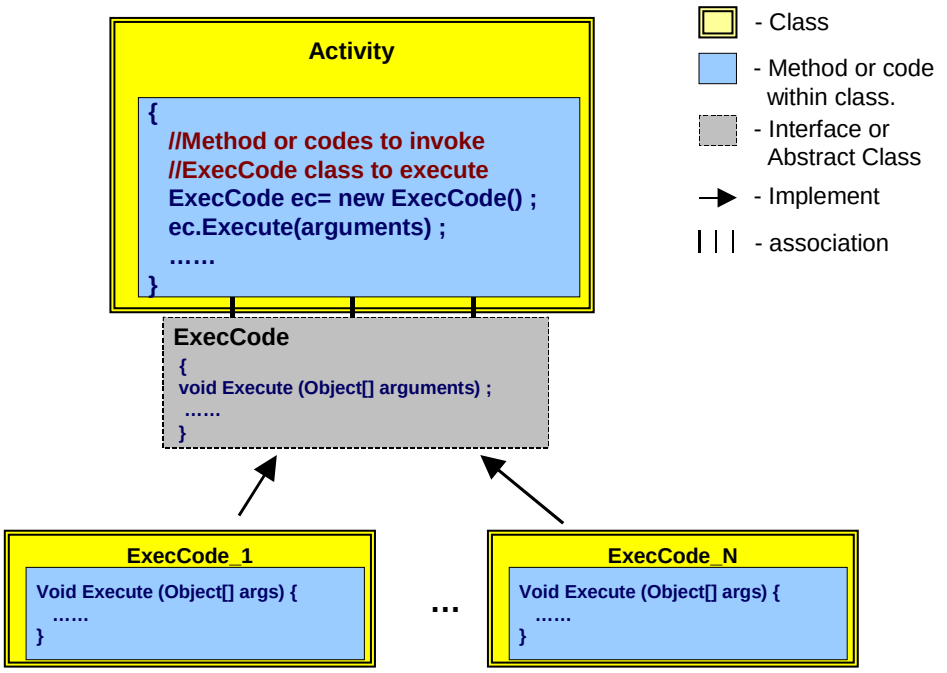

Figure 16: A more flexible design for Workflow Activity.

For a flow control purpose, the reusability of transition component is to be reused when meeting similar branches in different process or the change of process. However, object-oriented systems lack a procedural representation of control flow [8] [17]. The decomposition into classes typical of object-oriented architectures deemphasizes the control flow, distributing it among different objects. Therefore, an additional challenge of building object-oriented workflow architectures lies in providing abstractions that maintain an explicit representation of the control flow without violating the principles of good object-oriented design. An object-oriented workflow architecture must *provide abstractions* that enable software developers to define and enact how the work flows through the system [8].

4.3.1 User Interface

The reusability of user interface component is that a user interface can be generated

easily and developers do not need to care about the display problems of it when designing. As mentioned in section 3.3.1, we hope to provide a GUI editor for developers to design user interface. To provide a GUI editor, a class library of window components is needed. For instance, Figure 17 shows the user interface of a web-based workflow system. The participated user access the user interface by browser and Figure 18 shows the code fragment of this user interface.

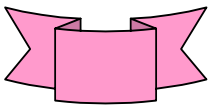

Figure 17: A screenshot of user interface in a web-based workflow

```
<P><font size=10>
Name:<%=name%>
</font></P>
<P><font size=10>
Email:<a
href="mailto:<%=email%>">
lwhiee@ice.ntnu.edu.tw</a>
<P>Picture:</P>
<img src=<%=image_path%>>
```

Figure 18: The snatch of the JSP code

In fact, the interlacing of HTML tags and JSP expression make us difficult to implement GUI editor and this code is hard to be reused. By contrast, the form of code of user interface shown in Figure 19 is much easier to be generated and is more suitable for GUI editor. Moreover, these window components may be displayed in different devices by invoking different components that implement different display protocols.

```
1. Label text1 = new Label(“Name:”) ;
2. text1.setValue(name);
3. text1.setSize(10);
4. Label text2 = new Label(“Email:”) ;
5. text2.setValue(email) ;
6. text2.setSize(10);
7. Image img = new Image(“Picture:”) ;
8. img.setImagePath(image_path) ;
```

Figure 19: The snatch of the source code of Figure 17 with another form.

In addition to the support of object-oriented programming language, the design of window components and the implementation of display protocols are the most important for user interface components to have the reusability we required.

4.3.2 Failure Recovery

The reusability of failure recovery component is that the failure recovery component does not need to be modified after the change of workflow process. That is, in an ideal situation, the failure recovery definition can still be used without modification after the change of process.

According to [13], the previous failure-recovery models have the following drawbacks - (a) They only allow specification of the ECP and compensation set of a failure in an activity in a static way before the workflow process is compiled. (b) The ECP and compensation set are specified by explicitly using the names (or some kind of identities) of activities. These models may cause the following problems – (1) inserting or deleting an activity may result in a change being required to the

specification of the ECP and the compensation set in other activities of the workflow process. (2) The ECP and Compensation set cannot be computed during the run-time of the workflow process. (3) Do not allow mutiple different failures in a single activity. Therefore, these models cannot meet our requirement.

[13] proposed the Workflow Failure Handling Language (WfFH) and a flexible failure recovery model to support WfFH language. The process designers may use WfFH language to specify how to compute the ECP and compensation set for the occurred failure situation. Because the ECP and compensation and some information of failure recovery definition are computed according to the execution results during the workflow process run-time, the change of process will not affect the failure recovery definition.

Although the object-oriented programming languages support the reusability of classes, the mechanism supports system to compute the data used for failure recovery, like the failure recovery model in [13], is critical for failure recovery component to reach the reusability we mentioned.

4.4 Process Definition Tools

As mentioned in previous section, the reusability of process definition tool is to be reusable among different workflow products. Object-oriented technology may promote the reusbility of objects or components. However, the reusability of process definition tool depends on the model language and meta-model adopted by workflow system. Different workflow systems could use the same process definition tool if they adopt the same model language and meta-model. But, as introduced at Section 0, several different workflow model languages have been developed but there are no

standards between them yet.

4.5 Distrubuted Workflow

As mentioned before, we believe that the platform-independent distributed components may have better reusability. Object-oriented programming language may be able to implement reusable components. However, these components are only reusable for homogeneous systems. The platform-independent feature should be provided by some platform-independent protocols or technologies, such as CORBA or Web-Service.

4.6 Persistence

The reusability of persistence component is that the persistence component can provide *database-independence* operations for workflow users. In fact, in addition to the support of object-oriented programming languages, a better design is necessary. For example, if we implement the operations of database manipulation within the persistence component, the persistence component will need to be modified when changing the original database system. As shown in Figure 20, the Persistence_A class cannot be reused immediately when we want to change the database system from MS SQL DBMS to Sybase DBMS. In fact, to provide database-independent operations, the data access object (DAO) pattern introduced by Java may be a good solution.

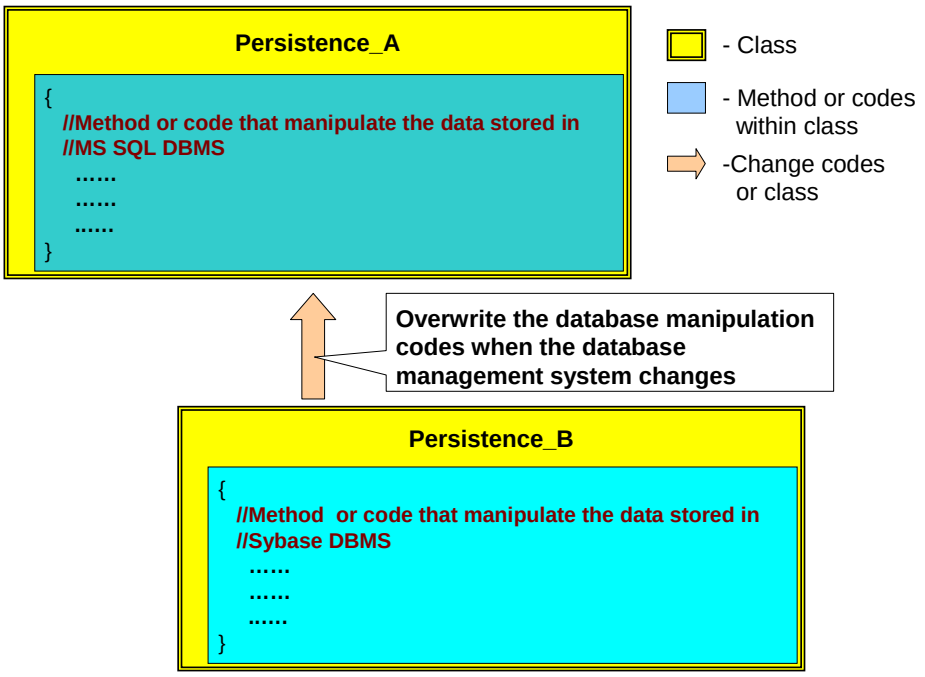

Figure 20: Change the implementation of storeData method by using inheritance.

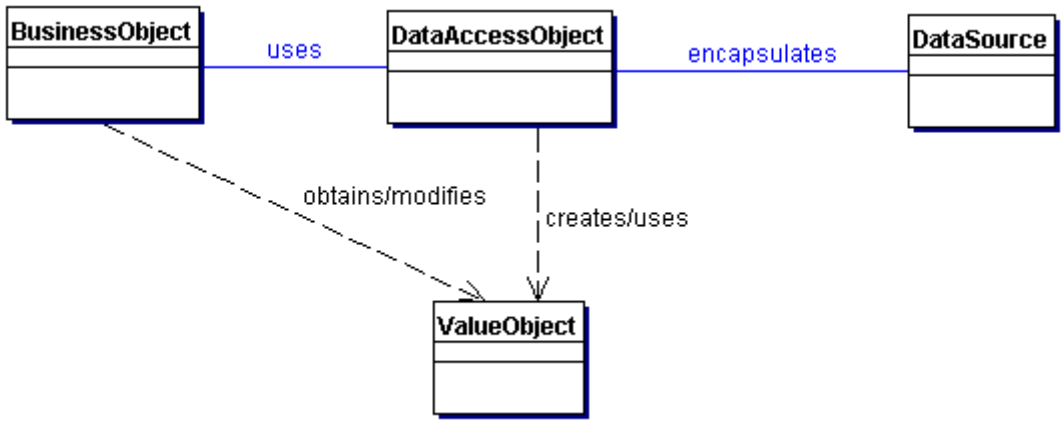

Figure 21: The class diagram representing the relationships for the DAO pattern

Figure 21 illurstates the relationship between classes in DAO pattern. `BusinessObject` first uses `DataAccessObject` to create `ValueObject`, and then, uses ValueObject to manipulate data (query, insert, delete and modify). Finally, `BusinessObject` uses `DataAccessObject` to store the data in `ValueObject` into `DataSource`. In this pattern, the business object (e.g. the workflow engine component) is responsible for the manipulation of data in

`ValueObject` and the operations to manipulate database systems are implemented in `DataSource` class. Therefore, the change of database system will not affect the business object. In fact, the use of database system is important to WfMS, the data related to system and process are kept in database. Therefore, every component in WfMS may need to manipulate database sytems. That is, the DAO pattern may not noly be used by persistence component but every component in WfMS.

As discussed above, we can use object-oriented programming languages and flexible design (e.g. a compositional design) for architecture to promote the reusability of WfMS components. However, because of some specific characteristics of WfMS, some components may have difficulties to reach the reusability we defined by using object-oriented programming languages only. For instance, to load activity classes and create acitivity instances automatically and flexibly, flow control of process and the reusability of user interface and failure recovery components. In the following chapters, we will introduce an object-oriented model for workflow management system and the designs to promote the reusability of components of WfMS.