UML 기반의 객체지향 프레임워크 모델링 기법
(UML-based OO Framework Modeling Techniques)

유 영 란 [†] 박 동 혁 [†] 김 수 동 [††]
(Young Ran Yoo) (Dong Hyuk Park) (Soo Dong Kim Kim)

요 약 컴포넌트 기반의 소프트웨어 개발 방법 (CBSD)에서 다양성(Variability)에 관한 연구는 컴포넌트의 재사용성을 향상시킬 수 있는 요소로, 그 중요도가 확대되고 있다. 주어진 도메인을 위해 개발된 컴포넌트가 다양성을 많이 지원할수록 개발된 컴포넌트가 적용될 수 있는 애플리케이션이 많아지기 때문이다. 그러나 컴포넌트가 많은 다양성을 지원하면 할수록 컴포넌트의 크기는 커지고, 개발 비용은 증가하기 때문에, 해당 컴포넌트를 이용해서 최적화된 시스템을 구현하는 일에 장애가 될 수 있다.

본 논문에서는 컴포넌트 개발 시에 부딪힐 수 있는 여러 형태의 다양성을, 먼저 성격에 따라 3 가지 유형으로 분류한다. 그리고 컴포넌트를 구현 시, 분류된 각 유형별로 적용이 가능한 기법들을 COM 컴포넌트 기반을 전제로 제안한다. 그리고 다양성의 추출부터 구현에 이르는 다양성의 분석에서 설계에 이르는 프로세스를 제공하는데, 이 프로세스는 컴포넌트의 개발 프로세스의 한 부분으로서 포함되어 적용이 가능하다.

Abstract The research of the variability gains more gravity in component-based software development, because it helps to extend the reusability of the component. A domain-specific component supports the more variability, the wider scope that the component can be applied to. However, the more variability included in a component, It makes the size of a component bigger, and the cost to construct the component is rises. As a result, this disturbs making an optimized system. In this paper, we classify the variability into 3 types, according to their features. And we propose some implementation techniques for each type based COM. Moreover, we also propose a process to analysis and design the variability with their artifacts, which includes some tasks from variability extraction to implementation of it. This proposed process can be applied as a part of the component developing process.

1. Introduction

As the component raises a principal factor in software engineering to increase the productivity and reusability of the software, the component-based software engineering (CBSD), including component extracting and implementation, component-based system construction, verification of component interfaces and other related techniques, raises significant issues in software engineering. Because the component is provided in a binary or source code form, it must be dependent on a specific software architecture. COM, CORBA and EJB are major component architectures that are broadly used, and they interest most of component developers. Especially, COM proposed by Microsoft is adopted by most of the commercial components, and makes application developers construct software system using binary typed components[6]. Although the COM is dependent on the operating system of a

† 학생회원 : 숭실대학교 컴퓨터학부
yuandyu@sclab.soongsil.ac.kr
dhpark@sclab.soongsil.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@computing.soongsil.ac.kr
논문접수 : 1999년 9월 13일
심사완료 : 2000년 1월 14일

hardware, owing to the marketability of Windows98 and WindowsNT that are used in the most of the personal computer, and rich case and development tools, it has its own unchallenged position among the component architectures.

To maximize the reusability of the component, the importance of extracting components from the domain is on the increase. One of the technologies for extracting component is commonality analysis, as domain engineering. The commonality analysis has focused their work on finding the commonality and the variability in family, is known as a domain. Identified commonalities are based of components and variabilities must be applies to each component for reusability extension.

In this paper, we propose a technique that implements the variability of component in COM architecture, and a process for extracting and implementing the variability. Section 2 describes a researches related to the COM and commonality analysis. Section 3 introduces some technologies of implementing variabilities. In Section 4, we define the terminology about the variability, and describe the process for implementing the variability based on COM. Section 5 provides an example using our approach in EC (Electronic Commercial) domain, and assesses and compares this work to related researches. Finally, in Section 6, we provide a conclusion of our work.

2. Related Works

2.1 COM

COM (Component Object Model) proposed by Microsoft is an object-based programming model designed to promote software interoperability. The softwares that participate in the interoperability may be an applications or components, even if they were written by different vendors at different times, in different programming languages, or if they are running on different machines different operating systems. To support its interoperability feature, COM defines and implements mechanism that allows applications to connect to each other as a software object[15]. To be a COM object that is a unit of software object, a developer must define an application programming interface (API) to allow creation of COM object for·use in integrating client applications or to allow diverse COM object to interact, and the COM object must adhere to this binary structure specified by Microsoft[9]. COM servers the connection a client and an object. However, once that connection is established, the client and an object communicate directly without having to suffer overhead of being forced through a central piece of API code as illustrates in Figure 1[15].

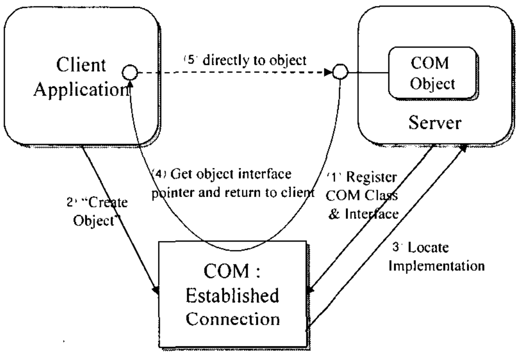

Client Application
(5) directly to object
COM Object
Server
(4) Get object interface pointer and return to client
(1) Register COM Class & Interface
(2) "Create Object"
(3) Locate Implementation
COM : Established Connection

Fig. 1 Connecting and communicating a client and a component in COM

In COM, four conditions are referred to successfully construct software system based on components[6]. Basic interoperability is the first condition. It is how developers can make their components that can interoperate with other components built by other developers. Version management is the second one, and it is how a component can be upgraded without change propagation. Third, language independence is how to communicate between components written in different programming languages. Last, communications by network, i.e., how can we make our own components that can be runed and interoperated by local and remote system, using one simple programming model [6]. To satisfy these conditions, some basic concepts are defined to provide structured foundation of COM.

- Binary Standard

Binary standard is defined for function calling between components to solve basic interoperability and language independence. This regards the interface that is independent from a type of internal implementation as fundamental communication means, without any defining about how one programming language associates with code standardization[4].

- Interface

To upgrade a component independently from other components and to manage the versions of component, once an interface is announced, the interface and its specification can not be modified any more[4]. That is, the COM interface is immutable.

- Local/Remote Transparency

From a viewpoint of a client, all component objects can be accessed through interface. If the target server exists in local system, then the client can directly access the interface of the component. Otherwise, the client can access the interface of the component through a proxy object that have same interface as one of the target component. From a viewpoint of server, all callings for interface methods of the component object are made through the interface. If the component object is a local server, the caller can be client. Otherwise, the caller can be stub object. The stub object provided by COM receives a remote method calling from proxy and translate it into method calling of the server component, as shown in Figure 2[6].

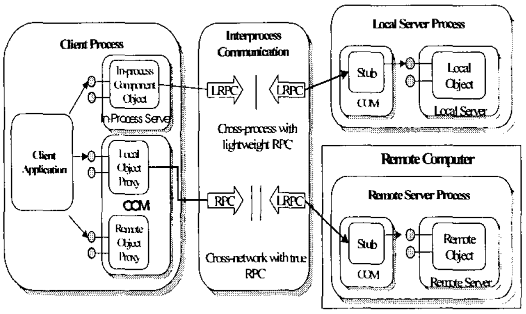


Fig. 2 Method callings between a client and a server

2.2 Commonality Analysis

Recently, due to increased interest in reusing software related technologies, that is, component software, framework, and etc, are critical issues to the industry and the academic. Among these technologies, commonality analysis is often used as a domain engineering technique. Commonality analysis is a domain engineering technique that emphasizes what is common and deemphasizes details among application family. Therefore, commonalities are requirements that hold for all family members[13]. By performing a commonality analysis, we can form and elaborate families.

To define a family, we need to formalize the terminology used to describe the family and to predict how family members will vary in order to identify the scope of the family. By formalizing the terminology, we can make communications among developers easier and more precise. And at the end it can be used as the guideline of finding reusable components that a reuser wants to use.

Commonality analysis provides three merits. That is, first, it provides abstraction. By grouping together a lot of classes under an abstraction, it helps decreasing the complexity of the design. Second, grouping by commonality analysis inherently leads to groups that are decoupled each other because each group has low commonality with the elements in other groups. Third, maintenance cost can be reduced by commonality analysis. The broader a commonality is, the longer the lifetime of a system is[12].

The variability means that how the family members is various to each other. A designer must consider not only the known variations but also the variations to occur in the future. The variablities that found in this way are categorized in accordance with the dimension of it, and the range of values of each variability is defined. Moreover, an analyst must specify when the value is bound to.

Figure 3 shows a process to define families. This process consists of two activities and three phases, that is, *plan*, *analyze*, and *quantify*. In analyze phase, common terminologies are defined, the commonality of domain is identified, and the variability is identified. Based on the result, open issues are identified. In quantify phase, the parameters of variations are defined.

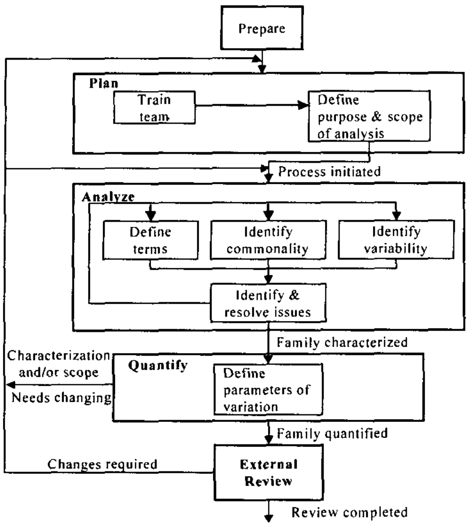


Fig. 3 A systematic process for defining families

2.3 Keepence and Mannion's Variability Modeling

Barry Keepence and Mike Mannion propose some model for designing the variability[16]. They classify the variability into 3 types, and propose pattern model per each variability type. They classify the variability into three types, that is, single discriminant, multiple discriminant, and option discriminant. Single discriminants are a set of mutually exclusive features, only one of which can be used in a system. Multiple discriminants are a set of optional features that are not mutually exclusive; at least one must be used. Last, optional discriminants are single optional features that might or might not be used. They have defined three patterns to model the three discriminant types[16].

The single discriminant can be modeled as an inheritance hierarchy in which generic features are modeled in a base class and specific features are modeled as subclasses. It is a single adapter pattern, as shown in. Figure 4[16].

The multiple adapter pattern models the multiple discriminant, like as Figure 5. The multiple discriminant can be modeled in the same way as a single discriminant; as an inheritance hierarchy, with generic features modeled in a base class and specific features modeled as subclasses. However, in this pattern, more than one subclass can be instantiated in any single system[16].

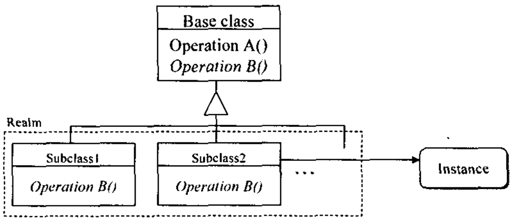


Fig. 4 Single adapter pattern

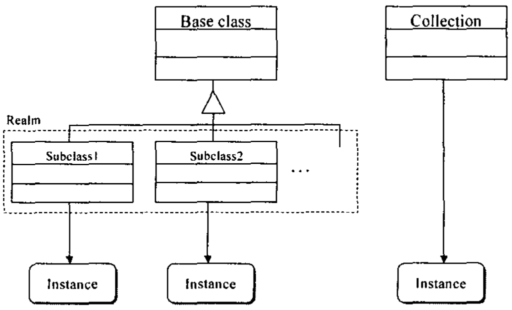


Fig. 5 Multiple adapter pattern

The option pattern models the option discriminant. When a feature is optional, it is modeled by creating two associated peer classes. It is not modeled optionality as inheritance, because inheritance cannot be optional. The associated classes must have a 0-1 relationship on at least one end. Figure 6 shows an example of it[16].

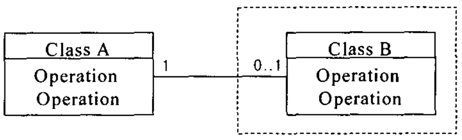


Fig. 6 Option pattern

They propose a good guideline to classify the variability by the implemented type of the variability. However, they do not support some techniques to implement the variability. And the classification lacks a consideration for type of the variability.

3. COM-based Method of Variability Implementation

3.1 Variability Classification

The variability can be classified three kinds, that is, the variability of attribute, logic, and workflow. The variability of attribute is one about the attribute that shows the information or state of a component. The variability of logic is one about the logic or algorithm in method of a component. Last, the variability of workflow is one about the workflow that is message flow among objects in a component. In this paper, each of the three kinds is called as VA, VL and VW. Generally, because the variability of workflow is not supported by most of the commercial components, we just deal with the former two types, in thus work.

In this paper, we propose some possible techniques to implement variability per each case. When component developer chooses a technique among proposed techniques, he(or she) must take care of its own merits and demerits, and factors that are a consideration to determine one technique among proposed techniques.

3.1.1 Variability of Attribute

- **VA1-Constant.** If the values or the ranges of the variability of property are determined before developing a component, we can make them to constant values of the component. Therefore, the binding time of these variabilities must be specification or compile time. For example, max_order_count = 99 in Order component is this case.
- **VA2-Method.** When an application want to change the values of the attributes of a component, the customization methods of API that the component provides can be called. Therefore, a value of these attribute can be dynamically modified. For example, set_OrderCount(int OrderCount); is this case.
- **VA3-DataFile.** By providing a data file that contains the values of the variabilities, when the component is initialized, the component reads the data file and sets the values per each attribute. In this case, if an user or reuser want to change the value of these attribute, they must change the contents of the data file. However, if there are many attributes to be customized, this technique reduces the API methods of the component. Table 1 shows the merits and demerits of the techniques that we proposed.

Table 1 Comparison among techniques for variability of attribute

Techniques	Merits	Drawback
VA1_Constant	-Simple code -Easy to use	-Limits the choice of reuser
VA2_Method	-Simple to implement a component -Dynamic customization by end user	-Needs more API for customization -The more variability, the more overhead at application and API of the component
VA3_DataFile	-Simplified interface -When the number of the variabilities is large, it is convenient to implement an application	-Requires an additional file -Not so dynamic (by end user)

3.1.2 Variability of Logic

- **VL1_StaticMethod.** If all choices of the variability of logic are determined by coding time, then, we can make these choices to methods. An application just calls the method having the logics with a parameter that determines what logic is chosen. However, new logic can not appended without breaking the component.
- **VL2_Method&ExtComponent.** To support providing a new logic to a component, we declare a required interface and additional method that delegates a calling to external component, in a component. Then reuser can use the method for the known logics with parameter by calling method set method, or delivery the pointer of the external component interface for unexpected logic by calling method *setOrder*. Before passing a pointer of external component to the target component, the reuser must make the external component, that is Component X in

Figure 7, with the same required interface. And then, any application calling the method DoIt, can obtain the proper logic by the predefined setting.

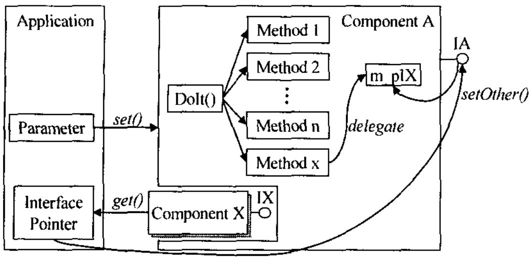


Fig. 7 Conceptual diagram of VL2-Method&ExtComponent

• **VL3_Class&ExtComponent.** The Classes for all the known logics and external component for possible one that have same interface are provided. Then, a reuser choose a class among classes provides by calling method set, or pass the reference of the external component to the component to set the variable by calling method *setOther*. The same functionality that has various logic is called by the variable, for example, *m_pIX*.

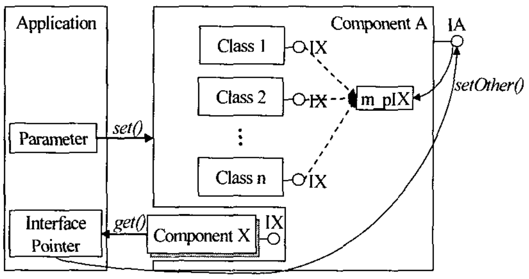


Fig. 8 Conceptual diagram of VL3-Class&ExtComponent

• **VL4_ExtComponents.** External components for all known logics and the possible ones are provided. First, we make external components having a same interface for all the known logics, and then we can add an external component for the unknown logic. In application, the pointer of the proper component is passed to the target component by calling method *set*, and then a method of the target component is called. The method delegates the message to each external component by pointer variable *m_pIX*.

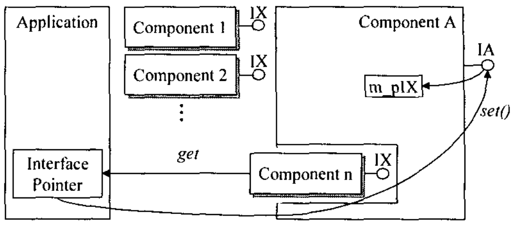


Fig. 9 Conceptual diagram of VL4-ExtComponents

Table 2 shows the merits and demerits of the technologies that we proposed. When we select only one technology among them, we must give a careful consideration the merits and demerits of them and choose one among them.

Table 2 Comparison among techniques for variability of logic

Techniques	Merits	Drawback
VL1_StaticMethod	-Light-weighted component -Easy to use	-Non-extendable
VL2_Method&Ext Component	-Extendable	-A speed difference between inner module and the outer component
VL3_Class&Ext Component	-Extendable -Variability that has its own properties and behavior	-A speed difference between inner module and the outer component
VL4_Ext Components	-Light-weighted component in case of heavy weighted variability	-Prone to run-time error due to lacking of static checking (semantic problem)

3.2 Criterion

• **Binding Time.** This means the time that the value(or logic or workflow) of the variability is determined. There are 4 kinds of the binding time, i.e., specification, compile, build and run time.

• **Performance.** This means the speed for customizing the variabilities and running the component.

• **Productivity.** The more the degree of difficulty of implementing the variability is, the lower the productivity for constructing a component is.

● **Size of Variability.** If the variability is given much weight in the total component, we can consider making the variability to an external component.

● **Flexibility.** The flexibility means that how many areas that the component can be applied. The more flexibility of component is, the more reusing of component is. However, too much flexibility means that the functionality of the component does not well-defined, that is, the flexibility of component is as much as possible, not more.

● **Convenience for Using.** The developed components are encouraged with convenience for using. Although, the functionality of a component is excellent, if the component cannot be used easily in an application, the reusability of the component is lowered.

● **Domain Requirement.** If there are some considerations in the requirements of domain and they are the major factor to determine a technique, we can not ignore them.

Among these factors, some factors can conflict with each other. Therefore, when one is emphasized, the others can be deemphasized. Following table shows the guides for choosing a technology among the provided technologies (Table 3).

4. Incarnation of Variability

Table 3 Factors and appliable techniques for variability of attribute and logic

Kinds of variability		Attributes			Logic			
Techniques / Factors		VA1	VA2	VA3	VL1	VL2	VL3	VL4
Binding Time	Compile	O			O			
	Run		O	O		O	O	O
Performance		O	O	O	O	O	O	
Variability Size		O		O				O
Productivity		O	O	O	O	O		
Flexibility			O	O		O	O	O
Convenience		O	O		O	O		

In this chapter, we propose a process for implementing the variability, based on COM. We assume that the analysis process of CBSD is accomplished, including commanality analysis, and focus on the design and implementation processes of variability. First, we define some terminologies for variability analysis, and then we suggest the design and implementation process. Because we concentrate on the implementation of variability, we do not consider all needed processes for developing a component.

Before implementing the variability, we must ceritify the variability to determine what variabilities are implemnted, because all of the variabilities are not subjects to customize. Some of the variabilities are setted to constant value or predetermined logic, and some of the variabilities are implemented to customize the variability, and the others are discarded on analysis and design time.

4.1 Terminology

In this step, we explain and propose some terminologies related variability modeling.

● **Family**: A set of similar work or application or equipment. It means a domain of application.

● **Family Member**: A member of family. It can be an application.

● **Commonality**: Something that is true for all family members.

● **Variability**: Something that is difference among family members. There are three kinds of variabilities, i.e., variability of attribute, logic and workflow.

● **Variability Sibling**: Some variability that have an interaction among variabilities. For example, variability A depends on variability B, i.e., if A has x value of variability then B must has y value, we call these two variabilities as a sibling.

● **Parameters of Variation**: The factor that determines the variability. In EC family, if each family member has its maximum order, the parameter of variation may be *OrderCount*.

● **Variant**: Each case of the variability. If there are three cases of variabilities, each case can be the variant.

- **Policy**: Each parameter or strategy that determines each variant. It is an abstract word and used to explain the process of customization
- **Customization**: The process that modifies a component to coincide with an application. The implementation of the variability is showed customization points.

4.2 Process

In our work, we just propose the process that connects with variability. This process can be included an analysis and design processes for component development as some tasks. In analysis phase, we identify the variability of each component. During designing the component, we assign the identified variability to each component, and then, determine an implementation technique for each variability.

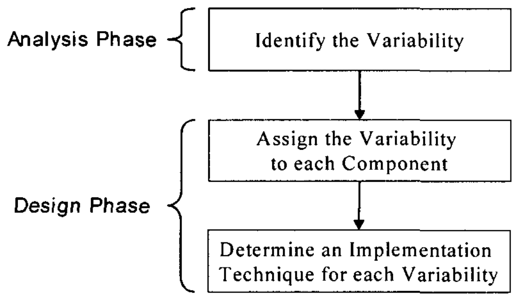

Analysis Phase
Identify the Variability
Design Phase
Assign the Variability to each Component
Determine an Implementation Technique for each Variability

Fig. 10 Process overview

4.2.1 Identify the Variability

Some variability is extracted from family. The variability can be the range of value that some functionality may have, or algorithm of the functionality, or workflow of some jobs in the family members. Extracted variabilities are classified, and the specification for each variability is defined. And then, we find the variability sibling from the extracted variabilities. Finding the siblings of the variabilities needs some more knowledge about family. Following Table 4 shows the matrix to find variability siblings. As shown in this table, variability V1 and V3 are the variability sibling, and variability V2 and V4 are one, also.

Sometimes, we can make all members of variability sibling into one variability. If a member of variability sibling is a variability of attribute and another is a variability of logic and workflow, then we can make them to one variability. However, if all members of the variability sibling are independent each other and they can be customized by client, and then we must not make them into one variability.

For each variability, we can define the parameters of variation and the range of values of each variability. And binding time and a default value are defined. The next Table 5 shows the example.

Table 4 Variability siblings matrix

Variability Variability	V_1	V_2	V_3	V_4
V_1			V	
V_2				V
V_3	V			
V_4		V		

Table 5 Parameters of variation

Parameters, Varaibility	Meaning	Type	Domain	Binding Time	Default
P_1:OrderCount, V_1	Number of order	VA	[1..99]	Compile	20
P_2:PaymentMean, V_2	Means of payment	VL	[cash, card, all]	Build	cash
P_3:ConfirmMail, V_3	Confirm mail	VL	[yes, no]	Run	yes
P_4:CancelPolicy, V_4	Policy of cancel order	VL	[part, all]	Specification	part

4.2.2 Assign the Variability to each Component

In this process, extracted and identified variabilities are assigned to each component that identified from the commonalities those are extracted at the domain analysis.

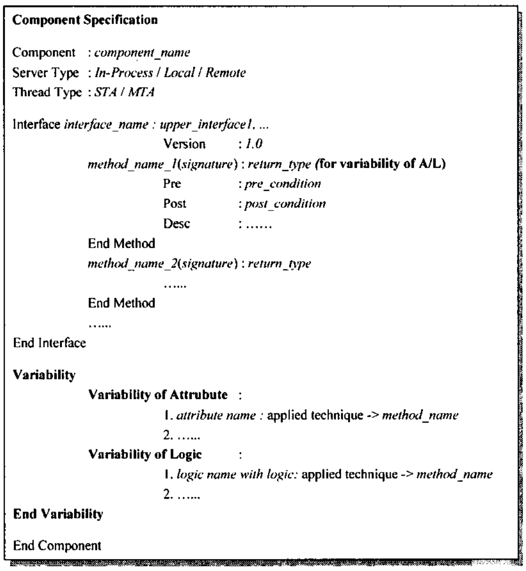
4.2.3 Determine an Implementation Technique for each Variability

In this task, a technique to implement per each variability is determined based on COM architecture. We can make Table 6 by using the guides proposed in chapter 3. After assigning a technique to each variability, we specify the variability in component specification. Figure 11 shows the component

Table 6 Implementation technique of each variability in a component

Component	Parameters, Varaibility	Type	Domain	Binding Time	Default	Applied Technique
CMP1	P1:OrderCount, V1	VA	[1..99]	Compile	20	VA1
CMP2	P2:PaymentMean, V2	VL	[cash, card, all]	Build	cash	VL1
	P3:ConfirmMail, V3	VL	[yes, no]	Run	yes	VL2
CMP3	P4:CancelPolicy, V4	VL	[part, all]	Specification	part	VL3

specification including variability specifiaction. Because a component specification is beyond of the scope of our work, we do not refer to this component specification.



```
Component Specification

Component   : component_name
Server Type : In-Process / Local / Remote
Thread Type : STA / MTA

Interface interface_name : upper_interface1, ...
                    Version     : 1.0
          method_name_1(signature) : return_type (for variability of A/L)
                    Pre         : pre_condition
                    Post        : post_condition
                    Desc        : ......
          End Method
          method_name_2(signature) : return_type
                    ......
          End Method
          ......
End Interface

Variability
          Variability of Attrubute  :
                    1. attribute name : applied technique -> method_name
                    2. ......
          Variability of Logic      :
                    1. logic name with logic: applied technique -> method_name
                    2. ......
End Variability

End Component
```

Fig. 11 Component specification with variability specification

5. Case Study and Assessment

5.1 Case Study

We make a selection EC (Electronic Commerce) domain as a family for our case study. EC is in the spotlight among applications using Internet. The priority of EC in the industrial is higher than before. The most typical case of EC is the advertisement. We focus on the Internet shopping mall. The main functionalities of shopping mall are searching goods, ordering, delivery, and so on. We especially pick up the *Ordering* among the functionalities, because the ordering is the most important part and has some variabilities to implement.

5.1.1 Identify the Variability

First, we define the family. In our case study, the family is EC domain, and family members can be 'Lotte Shopping Mall', 'Amazone Book Seller', 'Samsung Shopping Mall', and so on. We presuppose that the commonalities of this family are extracted and some components are identified from the extracted commonalities. Then we find the variabilities from our family, as shown in Table 7. In our case study, variability siblings are not found. Therefore, the matrix of variability siblings is omitted.

Table 7 Extracted variabilities

Variability ID	Type	Description
V_1	VA	The maximum number of order item is various. (9-99)
V_2	VA	The maximum day of waiting money coming in for orders is various. (1-7 days)
V_3	VL	When customer want to cancel order, according to a canceling policy in a component, the way to determine if the order can be canceled or not is different.
V_4	VL	When a customer inquiries the registered orders, the sequence of inquired orders is various.

5.1.2 Assign the Variability to each Component

Under assumption that the components are extracted and identified from the commonalities of the family, assign identified variabilities to components.

Table 8 Parameters of variation

Parameters, Varaibility	Meaning	Type	Domain	Binding Time	Default
P1:OrderCount, V_1	Number of order	VA	[1..99]	Run	20
P_2:WaitingDay, V_2	Number of waiting days	VA	[1..7]	Run	2
P_3:CancelPolicy, V_3	Policy of canceling order	VL	[impossible, conditional possible, possible]	Run	impossible
P_4:InqueryOrder, V_4	Sequence of inquiring	VL	[OrderNo, ItemId, DeliveryOrder]	Run	OrderNo

Table 9 Determined techniques to implement the identified variabilities

Component	Parameters, Varaibility	Type	Domain	Binding Time	Default	Applied Technique
OrderManagement	P1:OrderCount, V_1	VA	[1..99]	Run	20	VA3
	P_2:WaitingDay, V_2	VA	[1..7]	Run	2	VA3
	P_3:CancelPolicy, V_3	VL	[impossible, conditional possible, possible]	Run	impossible	VL3
	P_4:InqueryOrder, V_4	VL	[OrderNo, ItemId, DeliveryOrder]	Run	OrderNo	VL1

In this use case, we just extract the variability related to OrderManagement component; therefore the assigned table from the variabilities to components is omitted.

5.1.3 Determine an implementation Technique for each Variability

In this task, we determine the technique for the assigned variabilities per each component. Because each technique has its own merit and demerit, the requirement and objectives of an application must be considered to choose a technique. The following table shows the results of our case study.

5.1.4 Implementation Example

In this step, we implement two types of the variability, i.e., VA, VL, from the result of the previous case study. That is, the variability P1 and P3 are implemented by using the technique VA3_DataFile and the variability P3 is implemented by using the technique VL3_Class&ExtComponent. Each implementation result is as follows.

5.1.4.1 Implementation of variability of attribute

The type of the variabilities P1 and P2 is VA type. The techniques for variability of attribute are introduced in chapter 3. Among those techniques, we choose the VA3_DataFile techinque to set *Order-Count* and *WaitingDay* attributes of a component at once.

◈ Scenario

When a customer orders some goods, the quantity is limited to a value, which is different per each shopping mall. And the maximum days of waiting money for order are various.

◈ Solution

We make a data file that has values of those variables. And then, the variables are set by initial method of the component, when the component is loaded.

◈ Source Code

The *CreateOrder* method creates an order id and returns it. The *AddOrderItem* adds an item specified by *nItemID* to an order specified by *nOrderID*. The *SetChecked* method transforms the Not Paid state of the order specified by nOrderID to the state of Paid. The *Clean* method removes orders not paid till a due date. The *Init* method takes a parameter a file name that has in pairs an attribute's name and its value, reads that file and initializes the releated variables. The follwing figure shows the core of the source code.

```
interface IAddOrderCom : IUnknown
{
   HRESULT CreateOrder([out,retval] long* pOrderID);
   HRESULT AddOrderItem([in] long nOrderID, [in] long nItemID,
                        [out,retval] long* pResult);
   HRESULT SetChecked([in] long nOrderID, [out,retval] long* pResult);
   HRESULT Clean();
   HRESULT Init([in] BSTR filename);
};
```

Fig. 12 IDL of IAddOrderCom interface

```
class CAddOrderCom : public IAddOrderCom
{
        private:
        long     m_nMaxOrder;
        long     m_nWaitDays;

        ...
}
CAddOrderCom::Init(String filename)
{
        // Read 'filename' file
        // And then parse it into their field name and data
        // Then set m_MaxOrder and m_nWaitDays
        m_nMaxOrder = Parse(filename, "m_nMaxOrder");
        m_nWaitDays = Parse(filename, "m_ nWaitDays");
}
```

Fig. 13 Pseudo source codes of CaddOrderCom component

5.1.4.2 Implementation of variability of logic

The type of the variabilities P3 is VL type. Among those techniques for variability of attribute are introduced in chapter 3., we choose the VL3_Class&ExtComponent techinque to determine logic for canceling order.

◈ Scenario

When a customer wants to cancel the orders which has been ordered, the ways to determine whether the order can be canceled are various per each application.

◈ Solution

We make another interface to implement the variability, and classes that implement the interface to support predefined variability. And we add set method to set an undetermined policy to our pointer.

◈ Source Code

In this component, two interfaces are used. Interface IVL3CancelOrder has methods of which roles is setting the policy of canceling order and the method that transacts canceling order. Another interface IVL3CancelPolicy has a method that returns whether a canceling order is allowed. Figure 14 shows the definition of interfaces.

```
Interface IVL3CancelOrder : Iunknown
{
        HRESULT SetPolicy(long nPolicy);
        HRESULT SetOtherPolicy(IVL3CancelPolicy* pCustomCom);
        HRESULT CancelOrder(long nOrderID, long* pResult);
};
Interface IVL3CancelPolicy : Iunknown
{
        HRESULT CanBeCanceled(long nOrderID, long* pResult);
};
```

Fig. 14 IDL of IVL3CancelOrder and IVL3CancelPolicy interface

For VL3_Class&ExtComponent type of variability to be implemented, as presented in Figure 14, both of the classes within a component and an external component need to have a same interface. Therefore, they are substitutable each other. However, the inner classes are hidden in the component, and do not need the class factories for them. We create the class factories only for an external component and a main component, as shown in Figure 15.

```
class ATL_NO_VTABLE CVL3MainCom :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CVL3MainCom, &CLSID_VL3MainCom>,
        public IVL3CancelOrder
{ ..... }
class ATL_NO_VTABLE CVL3CancelPolicy1 :
        public CComObjectRootEx<CComSingleThreadModel>,
        public IVL3CancelPolicy
{ ..... }
class ATL_NO_VTABLE CVL3CancelPolicy2 :
        public CComObjectRootEx<CComSingleThreadModel>,
        public IVL3CancelPolicy
{ ..... }
class ATL_NO_VTABLE CVL3Custom :
        public CComObjectRootEx<CComMultiThreadModel>,
        public CComCoClass<CVL3Custom, &CLSID_VL3Custom>,
        public IVL3CancelPolicy
{ ..... }
```

Fig. 15 Definitions of the classes

As shown in Figure 15, class CVL3CancelPolicy1 and CVL3CancelPolicy2 are hidden whthin an order component, therefore, those classes do not inherit from CComCoClass which has codes responsible for creating a class factory. However, class CVL3Custom is an external component. Because class CVL3-CancelPolicy1, CVL3CancelPolicy2 and CVL3Custom inherit from the interface IVL3CancelPolicy, they are substitutable. As a result, the code excluding the code of setting a policy of canceling order is able to be independent of the type of the module used.

In Figure 16, the main source of class CVL3MainCom is represented. The method SetPolicy is invoked when a reuser sets a policy of caneling order. As shown, if that method is invoked taking the value 1 as the parameter, then the policy is impossible. And if it is invoked taking the value 2 as the parameter, then the policy is Conditional possible. Instead of it, if the method SetOtherPolicy is invoked taking the interface IVL3CancelOrder of an external component, then the policy is customized by the external component. At last, the method CancelOrder is used when a reuser actually implements the code of canceling order. It checks whether the requested order is possible to be canceled, and if possible, then remove that order from an Order DB

```
STDMETHODIMP CVL3MainCom::SetPolicy(long nPolicy)
{
        switch(nPolicy) {
           case 0:        break;
           case 1:        CVL3CancelPolicy1::_CreatorClass::CreateInstance(NULL,
                                   IID_IVL3CancelPolicy, (void**)&m_pCancelPolicy);
                          break;

           case 2:        CVL3CancelPolicy2::_CreatorClass::CreateInstance(NULL,
                                   IID_IVL3CancelPolicy, (void**)&m_pCancelPolicy);
                          break;

           default:       CVL3CancelPolicy1::_CreatorClass::CreateInstance(NULL,
                                   IID_IVL3CancelPolicy, (void**)&m_pCancelPolicy);
        }
}
STDMETHODIMP CVL3MainCom::SetOtherPolicy(IVL3CancelPolicy *pPolicyCom)
{
        m_pCancelPolicy = pPolicyCom;
        pPolicyCom->AddRef();
}
STDMETHODIMP CVL3MainCom::CancelOrder(long nOrderID, long * pResult)
{
        m_pCancelPolicy->CanBeCanceled(nOrderID, &fPossible);
        // possible to cancel an order transact actually a canceling process.
        if (fPossible != 0) {
                *pResult = 1; // successfully canceled order.
        } else {
                *pResult = 0; // keep from canceling this order.
        }
}
```

Fig. 16 Main source of class CVL3MainCom

5.2 Assessment

5.2.1 Comparison with other researches

There are not many researches on the customization of the component up to the present. Recently, commonality analysis includes the research of the variability. However, it just proposes an analysis process for finding and identifying the variability. And, in Keepence and Mannion's work, they classify the variability by their implementation type and propose a pattern-based model for them. However, they do not include implementation techniques. In our work, we propose some techniques to implement those variabilities based on COM, and propose the process to implement the variability in CBSD.

Table 10 Comparison between the proposed techniques and other techniques

Comparison items		Commonality analysis	Keepence and Mannion's work	Proposed techniques
Component Platform		Independent	Independent	Partially dependent
Appling to Component		No	No	Yes
Related with CBSD		Yes	Yes	Yes
Variability Classification	Implementation Type	No	Yes	Partially support
	Characteristics	No	No	Yes
Identifying Technique		Yes	No	Yes
Design Techniques		No	Yes	Yes
Implementation Techniques		No	No	Yes
Concrete Artifact		Partially support	No	Yes
Process		No	No	Support

Table 11 Comparison of proposed techniques among component platforms

Proposed Techniques		COM	EJB
Variability of attribute	VA1_Constant	Support	Support
	VA2_Method	Support	Support
	VA3_DataFile	Support	Support
Variability of logic	VL1_StaticMethod	Support	Support
	VL2_Method&ExtComponent	Support	Support
	VL3_ Class&ExtComponent	Support	Not support
	VL4_ ExtComponent	Support	Support

5.2.2 Comparison between other component platform

To find some reusability of our method and artifacts for other component platform, that is, EJB, we make following tables. Table 11 shows the comparison of proposed implementation techniques among existing component platforms. The VL3_ Class&ExtComponent, can not applied in EJB, because of technical differences to build a component. However, other techniques have no problem in EJB.

Table 12 shows the comparison of artifacts among existing component platforms. From the result, we can know that the all artifacts of our method can be also reused in EJB.

Table 12 Comparison of proposed artifacts among component platforms

Artifacts	COM	EJB
Variability siblings Matrix	Yes	Yes
Parameters of Variation	Yes	Yes
Implementation Techniques of each Variability in Component	Yes	Yes
Component Specification	Yes	Yes

6. Conclusion and Future Work

6.1 Conclusion

In this paper, we classified the variability into three types, i.e., the variability of attribute, logic, and workflow. And we proposed some tailoring techniques for the variability of attribute and logic, and a process to extract, identify and implement. We defined some terminologies related to variability implementation, and proposed a systematic approach to implement the variability when components are developed. The implementation techniques for the variability of workflow will be studied as our future work.

6.2 Future works

In this paper, we just propose the design and implementation techniques for the variability of attribute and logic based on COM. The applicability to other component platforms of proposed techniques is a part of our future work. And, the research of the variability of workflow has to continue.

참 고 문 헌

[1] Dale Rogerson, Inside COM, Microsoft Press, 1997.

[2] Guy Eddon, Henry Eddon, Inside DCOM, Microsoft Press, 1998.

[3] Desmond E. D'Souza, Alan Cameron Wills, Objects, Components, and Frameworks with, Addison-Wesley, 1999.

[4] Clemens Szyperski, Component Software, Addison Wesley, 1998.

[5] Don Box, Essential COM, Addison Wesley, 1998.

[6] Sara Willliams and Charlie Kindel, The Component Object Model: A Technical Overview , Microsoft Release, http://msdn.microsoft.com/library/techart/msdn_comppr.htm, October, 1994.

[7] Capt Gary Haines, David Carney, John Foreman, Component-Based Software Development / COTS Integration, CMU Software Technology Review, October, 1997.

[8] Alan W. Brown, Kurt C. Wallnau, Engineering of Component-Based Systems, 7-15. Component-Based Software Engineering:Selected Papers from the Software Engineering Institute. Los Alamitos, CA:IEEE Computer Society Press, 1996.

[9] Ed Morris, Emil Litvak, Component Object Model(COM), DCOM, and related Capabilities, CMU Software Technology Review, http://www.sei.cmu.edu/str/descriptions/com_body.html, June, 1997.

[10] Erich Gamma et al, Design Pattern : elements of reusable object-oriented software, Addison Wesley, 1995.

[11] Mary Kirtland, Interface and Component Design with COM, http://www.microsoft.com/com/presentations/default.asp, February, 1998.

[12] James O. Coplien, Multi-Paradigm DESIGN for C++, Addison Wesley, 1995.

[13] David M. Weiss, Commonality Analysis : A Systematic Process for Defining Families, Second International Workshop on Development and Evolution of Software Architectures for Product Families, February 1998.

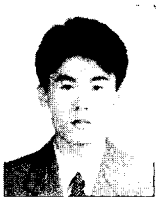
[14] James O. Coplien, Daniel Hoffman, and David Weiss, Commonality and Variability in Software Engineering, IEEE Software, 15(6):37-45, November/December 1998.

[15] Microsoft, The Component Object Model Specification Draft Version 0.9, Microsoft Press, October, 1995.

[16] Barry Keepence, Mike Mannion, Using Patterns to Model Variability in Product Families, IEEE Software, July/August, 1999.

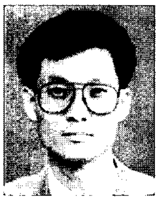
유 영 란

1993년 포항공과대학 전자계산학과 졸업. 1993년 ~ 1996년 현대전자 응용소프트웨어 개발팀 근무. 1996년 ~ 1998년 현대정보기술 의료사업부 근무. 1999년 ~ 현재 숭실대학교 대학원 컴퓨터학과 석사과정 재학중. 관심 분야는 CBD, 도메인 공학, ADL.



박 동 혁

1999년 숭실대학교 정보과학대학 컴퓨터학부 졸업. 1999년 ~ 현재 숭실대학교 대학원 컴퓨터학과 석사과정 재학중. 관심 분야는 객체지향 방법론, 컴포넌트 공학.



김 수 동

1984년 미조리 주립대학교 전산학과 졸업(학사). 1988년 The University of Iowa, 전산학 석사. 1991년 ~ The University of Iowa, 전산학 박사. 1991년 ~ 1993년 한국통신 연구개발단 선임연구원. 1994년 현대전자 소프트웨어연구소 책임연구원. 1995년 ~ 현재 숭실대학교 컴퓨터학부 조교수. 관심분야는 객체지향 개발방법론, 분산객체 컴퓨팅, 전자상거래 시스템.