JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 9, NO. 6, JUNE 2006(pp. 750-761)

A Design Technique of Component Framework Based on Framework Reference Model

Eun-Sook Cho†

ABSTRACT

As CBD technologies and researches have been matured, component framework as a larger reuse unit than component is being introduced. Especially issues related with adaptation and integration of components in CBD are being raised as a new research topic. The component framework is given as a solution to resolve these issues. However, current approaches don't suggest a sound and comprehensive reference model and development process applying reference model. In order to develop practical and stable component framework, reference model and concrete guidelines are essential elements. In this paper, we propose a generic reference model integrating existing reference models and a design technique of component framework based on it. Especially, we propose concrete and pragmatic guidelines such as how to design component framework architecture's view and style, how to design commonality and variability of component framework, how to design macro workflows among components, and so on. We believe that the proposed reference model becomes basis for component framework development, and the proposed design technique will support reliable and effective development of the component framework.

Keywords: CBD, Component Framework, Reference Model, Macro Workflow

1. INTRODUCTION

It is now clear that component-based development is going to be a new technology in software reuse. Various methodologies, tools, and platforms (EJB, .NET, etc) supporting CBD have been introduced. A component is a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces. However, current CBD approaches have some limitations. One of the limitations is overhead for the component consumers to identify and integrate the right components[4,5]. To overcome this difficulty of locating the right components, the concept of component framework or infrastructure was introduced [4]. The other is overhead for the component developers to reuse or assemble. Many component developers need larger reuse unit in practice. In that, a component framework is defined as a large reuse unit that contains a set of related components[2]. The granularity of component framework is larger than of a component. A component framework contains one or more components as well as relationships or workflows among components.

The concept of component framework was introduced as an application framework in[2], and a component infrastructure in PLSE[4]. However, the reference model of component framework has not been given enough research. Without a well-defined component framework reference model it is hard to have a good development process. The reason is that activities or instructions of development process are determined

※ Corresponding Author : Eun-Sook Cho, Address : (131-702) 49-3, Myeonmok-8 Dong, Jungnang-Gu, Seoul, Korea, TEL : +82-2-490-7562, FAX : +82-2-490-7396, E-mail : escho@seoil.ac.kr

Receipt date : Mar. 7, 2006, Approval date : June. 2, 2006

† Dept. Of Software, Seoil College

※ This work was supported by the 2005 Seoil College Research Fund.

according to the elements that are included in reference model. Therefore, a comprehensive reference model should be well defined.

Because the component framework includes several components, different types of component interfaces, component collaboration, variation points, connectors, and so on, a component framework reference model should cover these elements. However, current reference models do not cover all of these elements. As a result, some of the elements are not considered in the development process of component framework. Therefore, a comprehensive component framework reference model is needed.

Frameworks can not only speed-up component development and adaptation but they are also well suited to abstract and implement component collaboration. In order to maximize of component framework, we should reflect relationship and workflows related with component collaboration during component framework development process. However, current approaches deal with few macro workflows among components. To overcome this limitation, a comprehensive framework development process is needed.

In Section 2, we will introduce existing reference models of component frameworks and development methods. In Section 3, we describe a comprehensive reference model for component framework. Overall architecture of reference model and concepts of elements are described. We define a design method of component framework in Section 4. Section 5 describes traceability of artifacts. In Section 6, assessment results are described. We end this paper with concluding remarks.

2. RELATED WORKS

Several component frameworks have been proposed so far. However, very few of them explicitly focus on the precise definition of a reference model.

2.1 PuLSE(Product Line Software Engineering)

PuLSE methodology is developed for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts[7]. A reference model of PuLSE methodology does not cover all characteristics of component frameworks. It shows some elements of reference model such as control flow, control flow variation, component, logical component group, meta-condition, and one or two its variations. Therefore, this reference model does not provide the comprehensive set of elements. Beside, the precise definitions of these elements are not defined. This methodology focuses on how to customize or combine components or products in application development. Therefore, modeling guidelines for component framework are not defined adequately. For example, gaps occurring from component assembly are not analyzed and the ways to define and design connectors to fill these gaps are not considered.

2.2 SEI Product Line Framework

The approach identifies foundational concepts underlying software product lines and activities to be considered when creating a product line[6]. The listed practice areas comprise an extensive set of competencies and issues necessary to consider for successful adoption of product line based reuse. The viewpoint supports product line planning and management, rather than gives concrete instructions on implementing specific engineering tasks. Concrete guidelines to build framework, such as instructions related with characteristics of component framework; how to design connector types, how to represent macro workflows, and how to apply types of component interfaces in component framework building process, are rarely defined in this work. That is, instructions related with characteristics of component framework; how to design connector types, how to represent macro workflows, and how to apply types of component

interfaces in the component framework building process. Core assets are those assets that form the basis for the software product line. Besides, SEI product line framework gives no obvious definition on elements for solving the mismatch problem between interacting components and specific types of component interfaces.

2.3 The UML-F Profile for Framework Architectures

UML-F is an UML extension that supports working with object-oriented frameworks and allows the explicit representation of framework variation points[9]. A framework, UML-F assumes, is a collection of several fully or partially implemented components with largely predefined cooperation patterns between them. This framework implements the software architecture for a family of applications with similar characteristics, which are derived by specialization through application-specific code. However, elements are not explicitly identified in this model and no precise definition for the elements is suggested.

2.4 Reuse-Driven Software Engineering Process

Reuse-driven Software Engineering Business (RSEB)[8] is a use-case driven systematic reuse process. The RSEB integrates traditional object-oriented analysis and design with feature-oriented domain analysis (FODA) method. In this methodology, commonality and variability features are described through feature model. This process provides how to identify commonality and variability from feature model and use case model's variation points. However, this process mainly focuses on functional variability because it is based on use case model. Also, macro workflow between inter-components and the skeleton architecture description of component framework are not considered. Furthermore, the way to bridge occurring gaps in combining different components is not presented.

3. FRAMEWORK REFERENCE MODEL

In this section, we propose a comprehensive and practical reference model of component frameworks. A component framework consists of several elements; skeleton architecture, component interfaces, member components, inter-component relationships, and connectors as Fig. 1.

As shown in Fig. 2, a component framework should have one or more standard skeleton architecture. One or more interfaces and components that conform to the interfaces should belong to a component framework. A framework can have zero or more connectors that connect mismatching components. Three types of component interface and five types of connector are defined in this framework. A framework also has macro workflows, which is a sequence of dependency relationships over multiple components to carry out system level service.

Our framework supports variability mechanism. Variant is a possible solution (instance) for a variation point and, typically, a set of variants exists for a variation point[3]. In our framework, we consider three types of variability within the scope of a component as attribute, logic, micro workflow[11]. And for the framework level, we consider macro workflow variability.

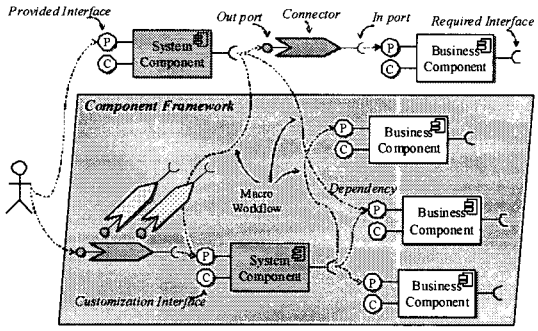

Fig. 1. Component Framework.

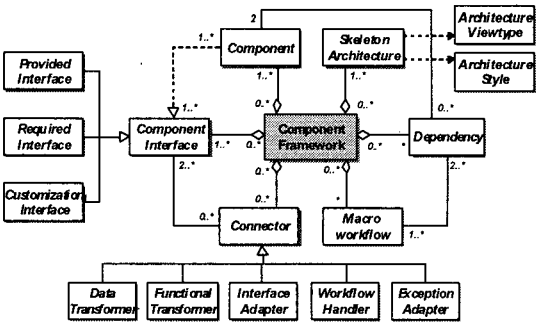

Fig. 2. Meta-model of a component framework.

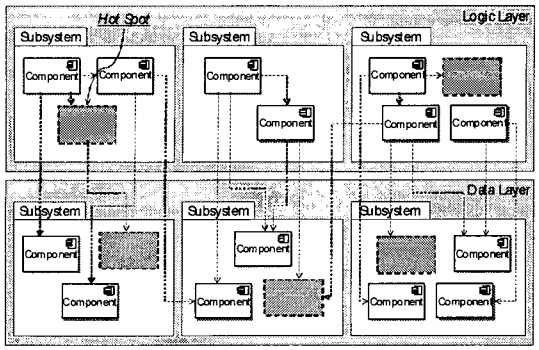

Fig. 3. Skeleton Architecture with Hot Spot.

3.1 Skeleton Architecture

A skeleton architecture is a general architecture of the target component framework. A skeleton architecture is distinguished from application architecture, because it is the architecture for the core part. Skeleton architecture provides standard reference architecture that will be instantiated for the target application. This is like a core set that needs to be enhanced to fit ones specific purpose through the incomplete part (hot spot) of the architecture as in Fig. 3.

Skeleton architecture is an important element that must be pre-designed because a component framework is a semi-completed application. We also use component framework to speed up the development of an application and pre-designed skeleton architecture supports this benefit of using a component framework.

3.2 Three Types of Component Interfaces

Current research works in CBD do not agree on the types of component interfaces and the names of the interfaces. This makes it difficult to define a framework reference model and framework development process. Hence, we define three types of interfaces as in Figure 1: provide interface, required interface, and customization interface.

Provide interface is a set of services provided by a component. These services are offered to the client in the form of operations. Required interface is a set of external services invoked by a component, i.e. a collection of external operation signatures used by the current component. Customization interface is a set of operations invoked to set variants to variation points. Intra-component variability is designed with this customization interface, and is used to tailor components for each application. Operations in a customization interface are mostly invoked only once per customization at deployment time.

3.3 Macro Workflow Model

A workflow is a sequence of method invocations between objects or components. A workflow within a scope of a single component is called micro workflow. A workflow outside the component and that acts as a mediator is called macro workflow. That is, a macro workflow is a sequence of method invocations over multiple components to carry out a system operation. Macro workflow is an important element because functionality serviced by framework is modeled through this element. Fig. 4 shows micro workflow and macro workflow inside a component framework.

3.4 Connectors

Locating a component that exactly matches to the specification of the component required is virtually impractical, because the component producer is often not the same as the component consumer. To overcome this difficulty, we need a mechanism that neutralize or mediate the unmatched aspects

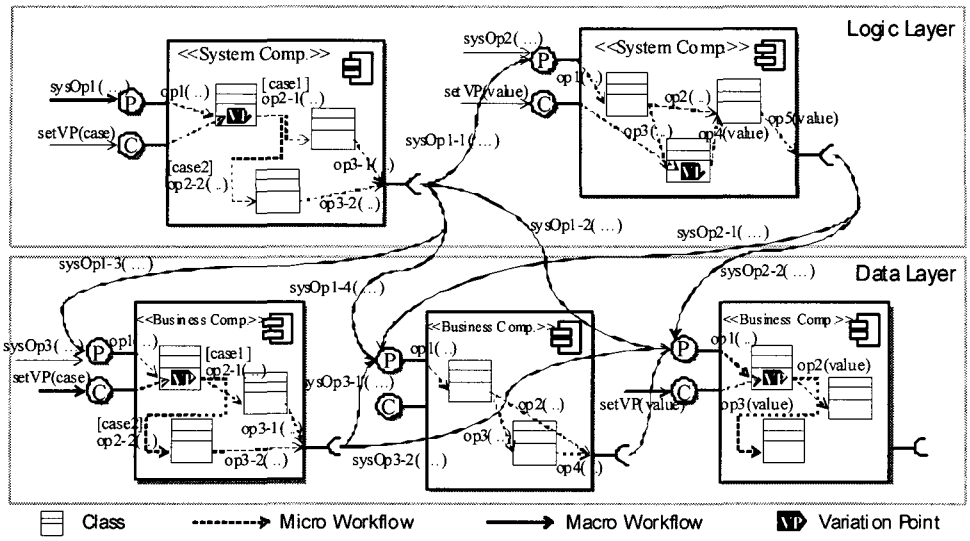

Fig. 4. Macro Workflow.

of the components. We call such software module or mechanism 'connector'[4]. We define some of symbols and terms to use in connector specification.

In Fig. 5, a target component is a candidate component that a client component wishes to use with a gap filling mechanism such as connector. We consider five types of connectors; data transformer, functional transformer, interface adapter, workflow handler, and exception adapter[10].

4. A DESIGN TECHNIQUE OF COMPONENT FRAMEWORK

In the previous chapter, a reference model of component frameworks has been proposed. In this chapter, we propose a systematic design technique to develop component frameworks that conform to the proposed reference model.

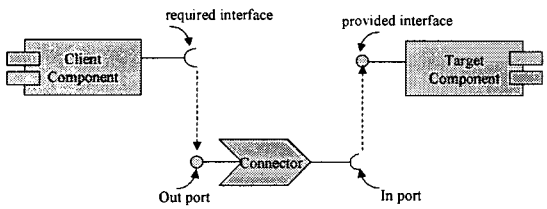

Fig. 5. Symbol Definitions for Connector.

4.1 Activity 1. Framework Requirement Establishment

Non-functional requirements are as important as functional requirement in developing component frameworks because frameworks include non-functional elements such as the generic architecture.

Step 1. Extract functional requirements.

Functional requirements are identified through use case modeling. Each use case can be common or similar among similar applications. Use cases describe system operation characteristics. Fig. 6 shows a use case diagram of rental management(RM) framework.

Step 2. Write use case descriptions for each use case.

We first write generic workflows for use cases. Use case descriptions are actor-driven or event-driven. In case of actor-driven, the first message flows should be initiated by actor. Besides, the first message flows should be initiated by system in case of event-driven.

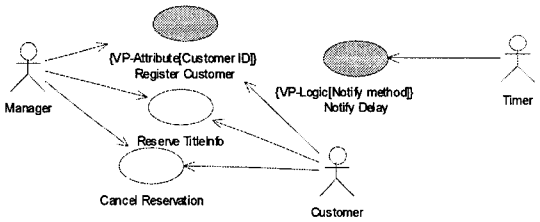

Fig. 6. Use Case Diagram of RM framework.

Step 3. Identify variation points.

Although functionality of common use case is the same, each family member can have different features. These different features are identified as variation points. Variation points can be variant attributes, variant logics, or variant workflows. Identified variation points are marked with stereotypes. Step 3 can be progressed in parallel because variation points can be identified during use case descriptions. In Table 1, RRN means Rental Registration Number, and SMS means Short Message Service. In case of 'Register Customer' use case, Customer ID is variation point. That is, Customer ID can be used differently according to the applications such as RRN or UnitID+Serial#.

Step 4. Extract non-functional requirements.

Framework occupies core part of an application. Therefore, framework's quality attributes have influence on the quality of overall application. These non-functional requirements affect selection of appropriate components a framework requires from available COTs(Commercial-Off-the Shelf) components. Non-functional requirements are identified from family members' non-functional requirements. Table 2 shows the non-functional requirement of RM framework. For example, there are availability, security, performance, reliability, and so on. Table 2 shows only availability part among non-functional requirements.

4.2 Activity 2. Skeleton Architecture Design (Reference Architecture)

A skeleton architecture of a framework is the generic and common structural views as well as behavioral views of the applications. Hence, the skeleton architecture is typically a fraction of the entire application architecture since the framework is only a large portion of the application rather than the entire application itself. The skeleton architecture makes frameworks quite distinct from components.

Step 1. Select architecture view type and styles.

Quality attributes of non-functional requirements are main inputs to select architecture view type and style.

An architect can either select one architecture view type or various architecture view types. If he selects various architecture view types, he can

Table 1. Variation Point Identification Table of RM framework

Use Case	Variation Point	Variation Type	Variants	Scope	Default
Register Customer	Customer ID	Attribute	{RRN, UnitID+Serial#}	Open	RRN
Notify Delay	Notify Method	Logic	{E-mail, SMS, Post-mail}	Closed	E-mail

Table 2. Non-functional Requirement of RM framework

Quality Attribute	Details
Availability	Provide 24-hour services
	Support global access
	Support a large number of concurrent users

merge various architecture view types into the component framework architecture. To select appropriate architecture view types and styles, we select appropriate architecture view types and styles by referencing values described in[1]. Also, additional architecture types and styles can be selected from stakeholders' focus. If stakeholders want to represent decomposition of overall system, decomposition style can be selected additionally. After selecting architecture view types and styles, an architect should describe design rationale for those.

Table 3 shows the architecture view type and styles for RM framework. In this example, view type is Module type, and architecture style is decomposition and layered style.

Step 2. Design elements to be included in skeleton architecture.

We identify elements to be assigned into skeleton architecture. An element can be a module, component, or subsystem according to architecture types and styles.

Step 3. Establish relationships among elements.

Identified elements are deployed in a part of architecture. There are relationships among elements. According to architecture view types and styles, needed relationships should be identified and designed. For example, there are relationships between components of higher level and components of lower level as in Fig. 7.

Table 3. Selected Architecture View Type and Styles for RM framework

View type	Style	Design rationale
Module	Decomposition	For availability, reliability, performance. And can also get modifiability.
	Layered	For security, reliability. And can also get modifiability.

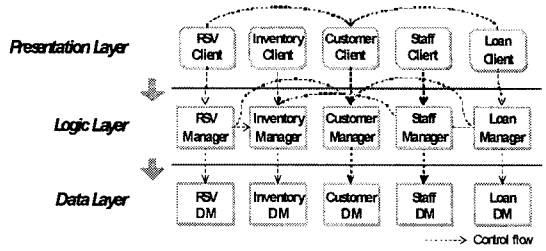

Fig. 7. Layered Style Architecture.

4.3 Activity 3. Locating Components

Step 1. Select components.

In order to select components, we see which components correspond to functional and non-functional requirements described in framework requirement specification.

First of all, functional requirements should be the same between available components and desired components. And then, interface specification should be satisfied. Third one is to consider non-functional requirements.

$A=\{A_1, A_2, \cdots, A_n\}$: A set of available components
$D=\{D_1, D_2, \cdots, D_m\}$: A set of desired components
$F=\{C_1, C_2, \cdots, C_o\}$: A set of selected components

If $D \subseteq A$, full match is generated. Here, $D \subseteq A$ means that functional requirements, non-functional requirements, and interface specification of D_j are equal to those of A_i. And then, the A_i is assigned into F.

Else if $A \cap D \neq \varnothing$ and $A - D \neq \varnothing$ or $D - A \neq \varnothing$, partial match is occurred. Here, partial match means that behavior, non-functional requirements, or interface specification is not matched completely. In this case, gap between two components, A_i and D_j, is generated. Therefore, solutions to fill this gap should be considered. It is executed by activity 5.

Else if $A \cap D = \varnothing$, mismatch is occurred. In this case, component specification should be prepared.

Step 2. Assign components.

Selected components should be assigned to right positions within a component framework. According to relationships of skeleton architecture, we connect identified components with each other. In this process, gaps among component interfaces can occur. These gaps should be filled with connector or new component development. Its guidelines are described in activity 5.

4.4 Activity 4. Workflow Design

Because a framework includes several components, we need to consider the inter-component message flows, i.e. macro workflows.

Step 1. Write design-level use case descriptions.

Use case descriptions of framework are different from traditional use case descriptions of application. Each use case includes one or more related components. Because workflows of each use case should reflect interface calls among components, design-level use case description is written by representing these interface calls.

Step 2. Describe interactions among components.

By going through workflows described in design-level's use case descriptions, we assign message flows to component's interfaces. The unit of interaction is not an object but a component; message flow is not between objects but between components. Variation points are also depicted in this diagram.

Step 3. Represent variant workflows.

By going through variant workflows described in design level's use case descriptions, we assign variant workflows to corresponding component's interfaces.

Step 4. Represent dependency relationships.

By tracing interaction diagrams, if one or more message flows among components are described, they should be depicted as dependency relationships among components. If there are no interfaces to be connected directly among components, gaps should be identified and filled through gap analysis in activity 5. Fig. 8 shows the dependency relationship between components in a component framework. As depicted in Fig. 8, message flows between components in a sequence diagram are mapped into dependency relationships in a component diagram.

4.5 Activity 5. Gap Analysis

The activity of gap analysis is to analyze the gaps identified through activity 3 and 4 and then to decide whether we develop new components or use appropriate connectors to fill the gap. In most cases, using connectors to fill the gap without the need to develop new components is more economical way. Determining the mechanism to fill the gap is in general a heuristic task and so defining a complete set of decision rules is infeasible and impractical. However, we suggest the following decision rules to guide the decision process in Fig. 9.

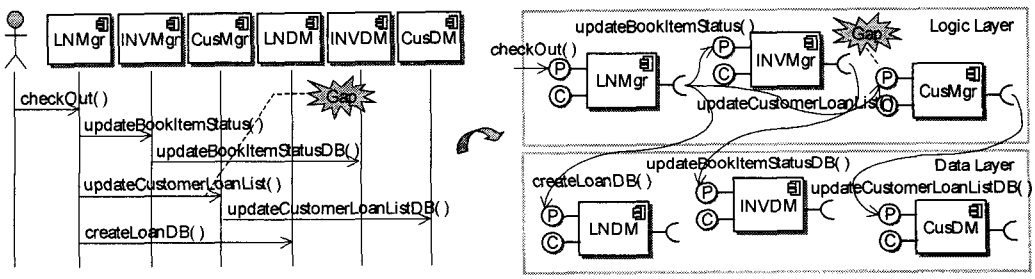

Fig. 8. An example of dependency relationships within a component framework.

4.6 Activity 6. Connector Design

A connector is a small software module that connects two components and possibly provides a designated functionality to remedy gaps.

Step 1. Define appropriate connector types.

In order to select appropriate connector, components' specifications selected from activity 3 are compared with workflows among components described in interaction diagrams. Also, we determine appropriate connector types for connectors identified from activity 5. As described in reference model of chapter 3, we consider five types of connectors[10]. Table 4 shows some of applicable connector types and roles of these types.

Step 2. Design connector's ports and algorithms.

In this step, we describe connector's ports and algorithms of connector according to identified connector type. Identified connectors have out port on one hand and in port on the other hand. According to connector type and role, connector algorithm may be different. We design connector algorithms according to connector type and role as shown in Fig. 10.

An example of connector design is depicted in Fig. 11.

4.7 Activity 7. Writing Framework Specification

A framework specification includes component's name, component's interface, workflows among components, variation points of a framework, and etc. This information can be gathered from the previous activities' artifacts.

The elements of the framework specification are as follows:

- Specification related with components contained in a framework; component's name, component's interface specifications, and component's constraints.
- Specification related with connectors; connector's name, connector's type, and connector's design specification.
- Specification related with overall framework's

```
switch(g)
case (contents of g are stable): select New Development;
                                 break;
case (granularity of g is simple & small): select connector;
                                 break;
case (cost/benefit of g is low): select connector;
                                 break;
case (side effect of quality attributes is large): select new development;
                                 break;
case (g is overlapped between different layers): select new development;
                                 break;
case (complexity of g is low): select connector;
                                 break;
default:
(Note)The g means an abbreviation of Gap.
```

Fig. 9. Decision Rules for Gap.

Table 4. An example of connector types.

Gap Type	Applicable Connector Type	Role
Gap1	Data Transformer	Reformat
Gap2	Functional Transformer	Add logic

```
T: Connector Type, C: Connector, R: Role
DT: Data Transformer, FT: Functional Transformer, IA:Interface Adapter,
WH:Workflow Handler, EA: Exception Adapter
if (T of C = = DT && R of C = = Reformat)
        reformat client.request.data type → target.receive.data type;
else if (T of C = = DT && R of C = = Transform)
        transform client.request.data.semantic → target.receive.data.semantic;
else if (T of C = = FT && R of C = = Logic change)
        change client.request.logic → target.receive.logic;
else if (T of C = = FT && R of C = = Add logic)
        add client.request.logic → target.receive.logic;
else if (T of C = = FT && R of C = = Reset)
        reset client.request.logic → target.receive.logic;
else if (T of C = = IA && R of C = = Separate)
        separate client.request.parameters → target.receive.parameters;
else if (T of C = = IA && R of C = = Combination)
        combine client.request.parameters → target.receive.parameters;
else if (T of C = = IA && R of C = = Change)
        change client.request.parameters.sequence → target.receive.parameters.sequence;
else if (T of C = = WH && R of C = = Change)
        change client.request.workflows → target.receive.workflows;
else if (T of C = = EA && R of C = = Change)
        change client.request.exception → target.receive.exception;
else
```

Fig. 10. Rules for Designing Connector Algorithm.

```
➢ Connector Name: LoanListIA
➢ Connector Type: Interface Adapter
➢ Connector Role: Separate
➢ Client Component: LNMgr
➢ Target Component: CusMgr
➢ Required Interface of Client Component: updateCustomerLoanList(cusID, bookItemID, dateTime)
➢ Provide Interface of Target Component: searchCustomer(cusID), updateLoanList(bookItemID)
➢ Out port: updateCustomerLoanList(cusID, bookItemID, dateTime)
➢ In port 1: searchCustomer(cusID)
  In port 2: updateLoanList(bookItemID)
➢ Algorithm:
  updateCustomerLoanList(cusID, bookItemID, dateTime)
  {   collect target.receive.parameters;
      separate target.receive.parameters;
      call searchCustomer(cusID) through In port 1;
      call updateLoanList(bookItemID) through In port 2;
      If success then return true else return false;          }
```

Fig. 11. An example of connector design.

architecture; framework's name, framework's architecture type and style, and framework's dependency relationships.
- Specification related with variation points; how many variation attributes, behaviors, and workflows are provided.
- Specification related with framework's hot spots. Hot spots are replaced by component's specification defined by framework developer.

5. ASSESSMENT

We proposed a component reference model in section 3. In Table 5, we compare the proposed framework with several other framework models introduced in section 2.1. Comparison criteria check if each framework supports elements with precise definition. SEI PLF means the SEI's Product Line Framework. In this table, criteria are elements related with component framework model.

As shown in Table 5, most of the frameworks do not consider detailed types of component interface, connector, variability and some frameworks do not support connector mechanism.(among existing frameworks, none support detailed types for interface, connector, and variability and only SEI PLF supports connector mechanism)

5.1 Comparing Processes to Develop Frameworks

In section 5.1, we defined the desirable properties of framework development process with justification in the proposed criteria. In Table 6, we compare our approach with representative framework engineering processes. Note that * means good and ** means superior. In Table 6, design criteria are elements related with component framework development process. In Table 7, design criteria means guidelines to design component framework reference model during development process.

As shown in the Table 6, all methodologies support the traceability among artifacts. Instructions for each activity are concretely suggested. For example, how to design component framework architecture, how to fill gaps between components, and how to describe variation points, and so on are presented specifically in the proposed process. Furthermore, because our process provides con-

Table 5. Framework model comparison table

Criteria \ Reference Model	PuLSE	SEI PLF	UML-F	Proposed Model
Skeleton Architecture	✓	✓	✓	✓
Component Interface	✓	✓	✓	✓
Components	✓	✓	✓	✓
Connector	-	✓	-	✓
Variability Concept	✓	✓	✓	✓
Detailed Interface Type	-	-	-	✓
Detailed Connector Type	-	-	-	✓
Detailed Variability Type	-	-	-	✓

Table 6. Coverage of compared methodologies for component frameworks

Design Criteria \ Methodology	SEI FSPLP	PuLSE	RSEB	Proposed Process
Traceability of Artifacts	**	**	**	**
Fine-grained Instructions	-	**	*	**
Stepwise activity or instructions	*	**	**	**
Formal instructions or concepts	-	-	-	*
Metric-based process	-	*	-	-

Table 7. Coverage of compared methodologies for component framework reference model

Methodology / Design Criteria	SEI FSPLP	PuLSE	RSEB	Proposed Process
Skeleton Architecture	**	**	*	**
Member Components	*	**	*	**
Three Types of Component Interfaces	-	-	-	**
Macro Workflow Model	*	*	-	*
Connectors	*	-	-	*

cepts and symbols clearly, proposed instructions and concepts can be adapted to formal specification and automatic transformation.

Also, we defined component framework reference model and its elements in Section 3.

In Table 7, we compare our approach with other approaches based on the degree of supporting these elements. As shown in Table 7, all existing processes do not consider three types of component interfaces, RSEB does not support macro workflow model, and PuLSE and RSEB do not support connectors. However, these elements are essential to build component framework. Our approach considers these elements as well as skeleton architecture and member components.

6. CONCLUDING REMARKS

We have presented a comprehensive reference model and design technique for component framework. The proposed reference model provides elements necessary to framework development. Especially, skeleton architecture, three types of components interface, and macro workflow model are proposed newly in our reference model. Because existing researches don't provide these elements, reusability or adaptability of component framework or component is very poor. However, developers can develop component framework easily and effectively by using proposed elements of component framework. Existing methodologies don't suggest concrete instructions or guidelines for component framework develop as well as component framework architecture. In this paper, we suggest a new design technique for component framework based on concepts of the proposed reference model. We believe that the proposed reference model and design technique make not only components reused effectively, but also applications built easily.

7. REFERENCES

[1] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architecture*, Addison Wesley, 2002.

[2] M.E. Fayad, and D.C. Schmidt, "Object-Oriented Application Frameworks," *Communications of the ACM*, Vol. 40, No. 10, pp. 32-42, Oct. 1997.

[3] L. Geyer, and M. Becker, "On the Influence of Variabilities on the Application Engineering Process of a Product Family," *LNCS 2379*, pp. 1-14, SPLC2 2002, San Diego, CA, USA, Aug 19-22, 2002.

[4] G.T. Heineman, and W.T. Council, *Component-based Software Engineering*, Addison Wesley, 2001.

[5] K. Whitehead, *Component-based Development: Principles and Planning for Business Systems*, Addison-Wesley, 2002.

[6] P. Clements, L. Northrop, et.al., "A Framework for Software Product Line Practice-Version 4.1," SEI, Sep. 2003, (http://www.sei.cmu.edu/plp/framework.html).

[7] B. Joachim, et.al., "PuLSE: A Methodology to Develop Software Product Lines," *The Symposium on Software Reusability'99*, Los

Angeles, May 1999.

[8] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process, and Organization for Business Success*, ACM Press, New York, June 1997.

[9] M.F. Fontoura, W. Pree, and B. Rumpe, "UML-F: A Modeling Language for Object-Oriented Frameworks," *14th European Conference on Object Oriented Programming* (ECOOP 2000), Lecture Notes in Computer Science 1850, Springer, pp. 63-82, Cannes, France, 2000.

[10] M. Hyen-Gi, and K. Soo-Dong, "Practical Connector Patterns for Designing Component Frameworks," *Accepted in Journal of Korea Information Science Society (KISS): Software and Applications*, pp. 43-53, Feb., 2004.

[11] S. Dong-Syeop, S. Sook-Kyeung, and K. Soo-Dong, "A Formal Model of Component Variability Types and Scope," *Journal of Korea Information Science Society(KISS): Software and Applications*, Vol. 30, No. 5, pp. 414-429, June 2003.

Eun-Sook Cho

received her B.E. degree in Computer Science from the Dongeui University at Busan of Korea in 1993. She received both M.S. degree and Ph.D. degree in Computer Science from the Soongsil University at Seoul of Korea in 1996 and 2000, respectively. She worked as a invited researcher in ETRI(Electronics Telecommunication Research Institute) at Daejon of Korea from 2002 to 2003. She worked as a full-time instructor in the department of Computer Science at Dongduk Women's University. Dr. Cho is a full-time instructor in the department of Software at Seoil College. Dr. Cho's research focus is software engineering, object-oriented modeling technique, software architecture, CBSE, MDA, software reuse, and web-service computing.