A Systematic Methodology for Adapting Software Components*

Soo Dong Kim and Hyun Gi Min

Department of Computer Science
Soongsil University
1-1 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
sdkim@ssu.ac.kr, hgmin@otlab.ssu.ac.kr

Abstract. Component-based development (CBD) is an effective paradigm for building software systems with reusable assets, where components often come in black-box form and expose interfaces while hiding internal details. Components are reused in developing various applications in a domain. However, the behavior provided by a component may not exactly match to the specification of an expected component. This is called a partial matching problem. Therefore, they have to be adapted for specific requirement. A component can be adapted in two ways; internal and external adaptation. For internal adaptation, the mismatch between a candidate component and the specification of an expected component can be resolved by some customization mechanism. However, if the component does not provide adequate customizability, it has to be adapted externally by using some external adaptation mechanism such as smart connector [1]. In this paper, we first propose a taxonomy for various mismatches during component acquisition. Then, we define a systematic process and practical instructions for identifying mismatches and adapting components.

1 Motivation

CBD is gaining popularity in both industry and academia as an effective reuse approach. CBD is accepted as an effective paradigm for building software systems with reusable assets. As the basic reuse unit in CBD, components often come in black-box form, only exposing well-defined interface while hiding internal details [2]. Since components are mainly for inter-organizational reuse, the behavior provided by components may not exactly match to the specification of the expected component. This is called *partial matching* problem. Therefore, they have to be adapted for specific requirements.

Components often come in binary and blackbox form to minimize the coupling between components and applications, and to protect intellectual property. Therefore,

* This work was supported by Korea Research Foundation Grant. (KRF-2004-005-D00172)

their source code and internal design cannot be modified by component consumers. This is especially true for commercial-off-the-shelf (COTS) components.

Components can be adapted in two ways; *internal* and *external*. For internal adaptation, the mismatch between a candidate component and the specification of the expected component can be resolved by using a pre-defined customization mechanism. This process is also known as *component customization*. If the component does not provide adequate customizability for the mismatch, then the component has to be adapted externally by using some external adaptation mechanism such as *smart connector* [1].

However, various issues in component adaptation are still remained open. The issue list includes a systematic way to identify and classify gaps between candidate components and expected components, a method to select the most optimal adaptation technique, and a process to apply the selected adaptation technique.

In this paper, we first propose taxonomy for various types of mismatches in component acquisition. Then, we define a process and instructions for systematically adapting components for target applications. By using the proposed framework, we believe that candidate components can be either more systematically customizable or adaptable, greatly increasing the applicability of components.

2 Related Work

Kim's work establishes a theoretical foundation on variability in CBD [3]. Variability is classified into five types; *attribute logic*, *workflow*, *interface* and *persistence*. The scope of variability is classified into *binary, selection,* and *open*. Essential variability-related terms are defined such as *Variation Point (VP)*, *Variant*, and *Variability*.

Catalysis method provides two customization techniques: *Inheritance and Template (IT)* and *Polymorphism and Forwarding (PF)* for implementing variable functionality [4]. With *IT*, generic methods are declared in a base class, and are overridden in derived classes. Effectively, a member-specific variant such as logic can be realized in a derived method. With *PF*, a subclass implements an abstract class by providing member-specific variant in virtual methods, and an instance of a subclass substitutes an object of its superclass.

Keepence and Mannion's work suggests three patterns for variability design; *single adapter*, *multiple adapter* and *options* patterns [5]. In *single adapter*, generic features are modeled in a base class and specific features are modeled in subclasses. Only one subclass can be instantiated in any single system. *Multiple adapters* are similar to single adapter, but more than one subclass can be instantiated in any single system. In *options* pattern, two associated peer classes are created to realize a variation. Keepence's work suggests three types of variability mechanism.

Anastasopoulos and Gacek's work identifies various customization methods; aggregation/delegation, conditional compilation, dynamic class loading, dynamic link libraries(DLL's), frames, inheritance, overloading, parameterization, properties and

static libraries [6]. Among the proposed techniques, aggregation/delegation, dynamic link library and parameterization methods can be applied to blackbox components.

Wrappers provide a simple abstraction that hides layers, and they simplify the task of programming [7]. A wrapper is a type of software "glueware" that is used to attach other software components together. Wrappers can be utilized to present a simplified interface, to encapsulate diverse sources so that they all present a common interface, to add functionality to the data source, or to expose some of the data source's internal interfaces.

Connector is an essential element of software architecture [8], and it is used to inter-connect components in a framework. A connector imposes role-specific constraints on the ports that it connects and can be refined to particular interaction protocols that implement the joint action.

3 Partial Matching Problems in Component Acquisition

3.1 Taxonomy of Partial Matches

Since components can potentially be used for various products, the behavior provided by candidate components may not exactly match the specification of the components required by various component consumers. This is a well known problem in component acquisition, but there can be different types of partial matches. It is a prerequisite to classify all possible partial matches in order to define more specific and detailed adaptation instructions.

To derive types of partial matches, we consider the building blocks of software components. A component in most current component reference models consists of classes, workflows among the classes, and interfaces. A class, in turn, captures attributes and methods. Some classes are persistent meaning their attributes have be persistently stored and managed. Hence, the five building blocks of software are identified; attribute, logic, interface, workflow, and persistency [3].

Table 1 shows the taxonomy for all types of partial matches. Let *CComp* be a candidate component and Let *DComp* be an ideal component desired by component consumers. As in Table 1, we identify 5 classes of partial matches; partial matches based on *attribute, functionality, interface, workflow,* and *persistency*.

Three types of partial attribute matches may occur on attributes; i) Some attributes required by the component consumer are not supported by CComp ii) Some attributes supported by CComp are not required by component consumers. iii) Data type of attribute varies between CComp and DComp. In the case of functionality partial matching problems, two types of partial functionality matches may occur on functionality; i) CComp should be appended with some additional functionality for DComp ii) The extra functionality of CComp should be disabled for DComp. In the case of interface partial matching problems, CComp has a provided interface which consists of

function signatures and their semantic descriptions. If a function satisfies the behavior required by DComp but its signature does not match the signature required by DComp, then there is a mismatch problem on the interface. A mismatch may occur on function name, types of input/output parameters, the ordering of parameters and return type.

Table 1. Taxonomy of Mismatch Problems

Partial Match	Types of Partial Match
Attribute	Some attributes required in DComp are not presented in CComp.
	Some attributes presented in CComp are not required in DComp.
	Data type of attribute varies between CComp and DComp. *Ex) Integer* vs. *String for AccountID in banks*
Functionality	Some functionality required in DComp is not provided by CComp.
	Some functionality presented in CComp is not required in DComp.
Interface	Name of operation varies between CComp and DComp. *Ex) addItem( ) vs. createItem( )*
	Type of parameter or a return type varies between CComp and DComp. *Ex) integer vs. float for a parameter* *Ex) Integer 0 or 1* vs. *Boolean* for a return type
	Value range of a parameter or a return type varies between CComp and DComp. *Ex) 0..1* vs. *0..100 in* percentage for *interest rate* in banks
	Order of parameters for an operation varies. *Ex) F(name, age, address)* vs. *F(name,address,age)*
Workflow	Operation of workflow type required in DComp is not provided by CComp. *Ex) A.F1()→A.F2()* vs. *A.F3() which includes F1() and F2()*
	Order of invocations in a workflow varies between CComp and DComp. *Ex) F1( )→ F2( ) → F5( ) vs. F5( ) → F1( ) → F2( )*
Persistency	Persistent database model employed varies between CComp and DComp.Ex) *Text File* vs. *Relational Table*
	Schema for relational tables varies between CComp and DComp. *Ex) Unification, Horizontal and Vertical Partitioning* [9]

In the workflow, the order of invocations in a workflow of CComp is not satisfied by the component consumer. Therefore, the order should be reorganized. The workflow has partial mismatching problems. In persistency partial matching problems, CComp has entity classes that have relationship among classes. The classes are needed for mapping objects to relational tables. The relational tables are various designed. Therefore, schema for relational tables varies between schema of CComp and schema of legacy systems.

3.2 Internal and External Adaptations

A component can be adapted in two ways; *internal* and *external* adaptation as in Fig. 1. For internal adaptation, the partial matches between a CComp and the specification of DComp can be resolved through the pre-defined customization interface of CComp.

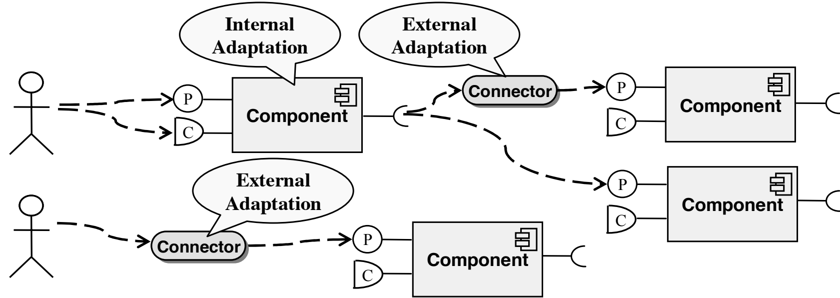


Fig. 1. Two ways of Adaptation Technique

Internal adaptation is applied by component customization. *Customization* is a task to set variants into variation points inside components using customize interface or mechanism [10]. The effect of customization remains inside the component. The variation points pre-defined for variation are filled by variants as in Fig. 2. The customer selects pre-implemented variants in a component or plugs in new variants for the application through customize interfaces to accept variants.

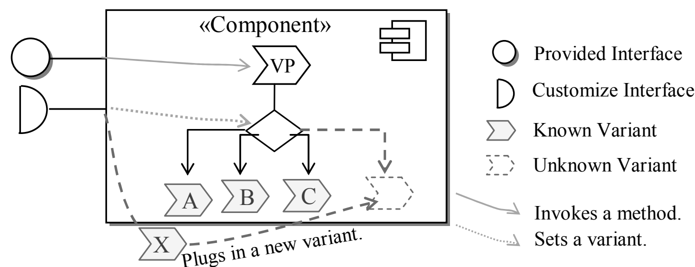


Fig. 2. Mechanism of Customizable Components

External adaptation uses *smart connectors* [1] that resolve mismatch problems among components. A smart connector is an external module that sits between the component consumer and the component and mediates the partial matches between them as in Fig. 3. Hence, the connector mechanism does not alter any internal part of the component. The smart connector has a transformation rule to resolve partial matches between components that have different functionalities, interfaces, data value ranges, and workflows. The *Smart connector* mechanism is used to adapt components without modifying the components themselves.

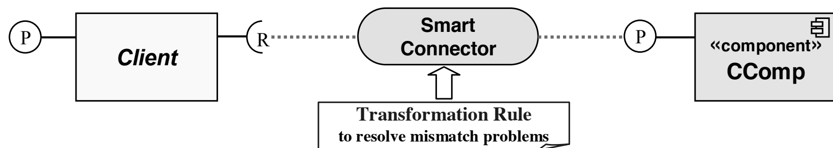


Fig. 3. Mechanism of Smart Connectors

4 The Process and Instructions

In this section, we present the overall process which consists of four phases as shown in Fig. 4, and we give instructions to carry out the activities.

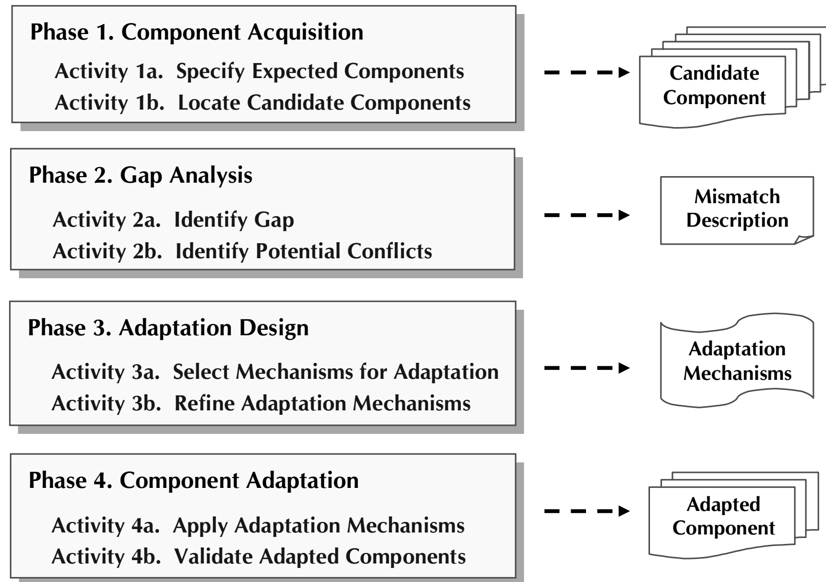


Fig. 4. The Process and Artifacts

4.1 Phase 1. Component Acquisition

This phase is to acquire suitable components for a target application.

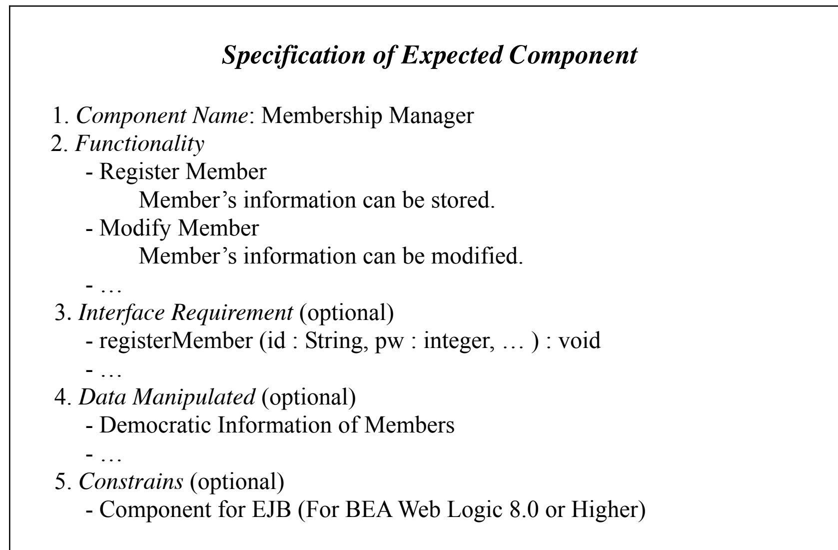
Activity 1a. Specify Expected Components

To locate the right components for a target application, the requirement for expected components should be described. Activity 1a specifies the requirement of expected components. This specification should include four parts; expected functionality, specific interface required if any, data manipulated, and constraints such as programming language, component platform, quality attribute, etc. An example of an expected component specification is given in Fig. 5.

Activity 1b. Locate Candidate Components

To describe and acquire the list of suitable candidate components, activity 1b searches and verifies candidate components for the target system. First, component customers survey component markets to get compatible component lists. The candidate component lists that can be used in the same domain are searched.

Second, the customers do some initial checking to confirm the compatible component, but detailed testing may not be required. Some COTS components are not opened before consumers buy them. Therefore, if we can explore them, the customer explores the searched initial component lists. The candidate components are explored by conceptual mapping functional and non-functional requirements such as component platform, component sale price, etc. While exploring, the number of candidate component lists are decreased. As a result, the customer can find some compatible components.



Specification of Expected Component

1. *Component Name*: Membership Manager
2. *Functionality*
   - Register Member
     Member's information can be stored.
   - Modify Member
     Member's information can be modified.
   - …
3. *Interface Requirement* (optional)
   - registerMember (id : String, pw : integer, …) : void
   - …
4. *Data Manipulated* (optional)
   - Democratic Information of Members
   - …
5. *Constrains* (optional)
   - Component for EJB (For BEA Web Logic 8.0 or Higher)

Fig. 5. Template for the *Specification of Expected Component*

Finally, the candidate components are verified. They conform to its specifications. There are risks of partial matching between the component specification and the real characters of the component. While verifying, faulty components may be rejected on the candidate component lists.

4.2 Phase 2. Gap Analysis

This phase is to identify gaps between the requirement specification for the target system and components. If a candidate component provides a limited applicability and customizability so that it does not completely satisfy the functionality needed, then a component consumer cannot reuse the component in application development. We call it a *partial matching* problem [1] in component acquisition.

Activity 2a. Identify Gap

The purpose of this activity is to identify the gap between expected components and candidate components. This activity identifies partial mismatch problem about

attribute, function, and workflow categories. The input artifacts are the expected components or requirement specifications of the target system. The outputs are partial mismatch problems information in candidate components.

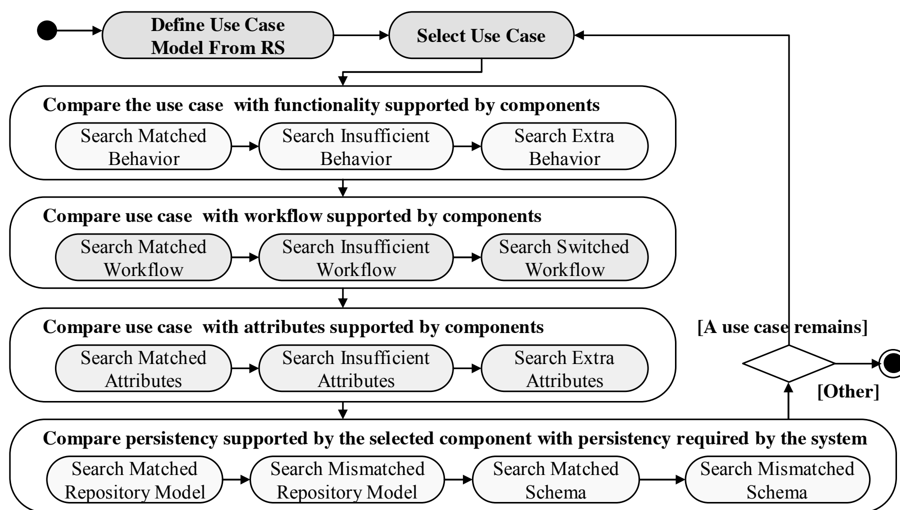


Fig. 6. The Process for Identifying Partial Match Problems based on Use Case

The taxonomy of partial match problems in Table 1 is used for this activity. First, the requirement specifications are refined. This step defines the use case model that includes a main flow, alternative flows, exception flows, and scenarios. Second, a use case is selected to find compatible components as in Fig. 6. Third, the selected use case is compared with functionality of candidate components. Sufficient behavior of the candidate components is identified. The partial mismatch problems of the candidate components are also identified. The candidate components have insufficient behavior and extra behavior. The extra behavior raises side effects and reduces performance of the target system.

Fourth, the selected use case is compared with the workflow of candidate components. Sufficient workflow of the candidate components is identified by flows and scenarios of use case. The partial match problems of the candidate components are also identified. The candidate components may need new workflow and have extra workflow.

Fifth, the selected use case is compared with attributes of candidate components. Each attribute is needed while the component performs the use case. Components should manage all attributes needed by the target system. The supported attributes in the candidate components are identified. The partial match problems of the candidate components are also identified. The candidate components may need more attributes and have useless attributes. Finally, degrees of match such as perfect match (FULL), partial match (PARTIAL), and no match (NONE) are identified.

Activity 2b. Identify Potential Conflicts

This activity identifies potential conflicts among selected components. Components that have inter-relationships are assembled to build applications. Components have *provided* and *required* interfaces. The provided interface specifies the services provided by a component and it is invoked by other components or client programs at runtime. The required interface specifies external services invoked by the current component, i.e. a specification of external services required by the current component [11]. By specifying the required interface for a component, we can precisely define the services invoked by the current component.

If components may not completely satisfy required interfaces expected by other components, the components cannot be used because a component is assembled with other components. The input artifacts are required and provided interfaces specification of candidate components, requirement specifications of the client program.

Components have provided interfaces that consist of function signatures and their semantic descriptions. If a function satisfies the behavior required by other components, but its signature does not match the signature required by other components, then there is a mismatch problem on the interface as in Fig. 7.

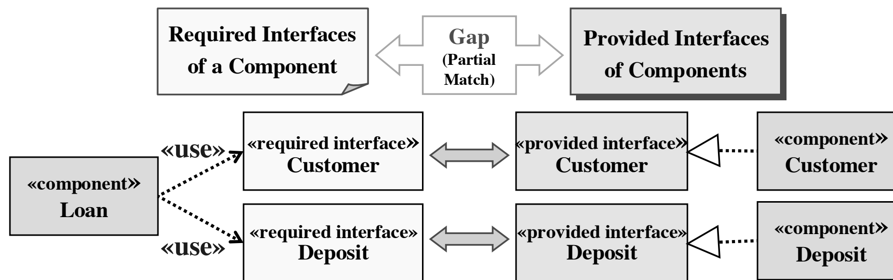


Fig. 7. Gap between Required Interface and Provided Interface of Components

A required interface and a provided interface have a key role in identifying partial match problems between components as in Fig. 8. First, the required interface of one of the candidate components is selected to find gaps with a provided interface of the component required by them. Second, the signature of a selected required interface is compared with the signature of a provided interface. Mismatch problems such as operation name, type of parameter, ordering of parameters, and return type can be identified.

Third, the semantic of a selected required interface is compared with the semantic of a provided interface. Potential problems are that the signature such as data type is correct but the data semantic such as data range, data meaning is incorrect. Programming compilers cannot find these problems.

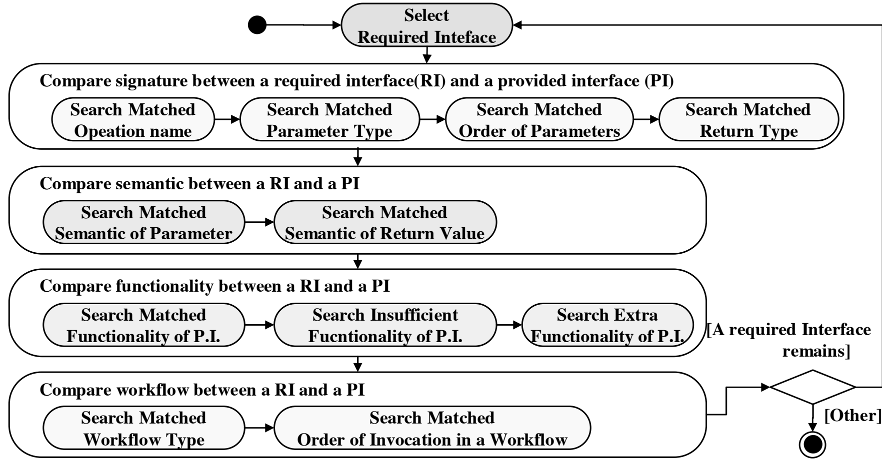


Fig. 8. Process for Identifying Partial Match Problems between Components

Fourth, the partially mismatching problems about functionality are identified. When a provided interface satisfies most of the functionality that the required interface expects but partially lacks, there are two cases when comparing the provided and the required functionality. Functionality is determined by the set of all functions in a component. Let a predicate *Fn(DComp)* be the functionality provided by *DComp* and *Fn(CComp)* accordingly. In case i), the extra functionality of *COMP* should be disabled for *DComp*. In case ii), *COMP* should be appended with some additional functionality by a connector.

Table 2. Specification for Describing Partial Match Problems

Use Case	Item	Element	Descriptions
Reserve a Room	*Requirement Specification*	Function	Reserving a vacant room Validating customer's SSN when a customer reserves a room.
	Candidate Component	Provide Interface	*reserveRoom (roomNum:String, customerID:String) : boolean*
			This operation is to reserve a room using a room number and general customer ID.
		Function	Reserving a vacant room
	Mismatch Problems	Sufficient Function	This CComp supports reserving a vacant room.
		Insufficient Function	This CComp does not support validating customer's ID

Fifth, problems about workflow are found. If provided interfaces have dependency with some ordering, the clients of them should call the provided interface through the ordering. Workflow is a sequence of method invocations among components to carry out a function in an interface. Workflow mismatch is distinguished from functional mismatches; workflow mismatch is determined by examining the orders of multiple

method invocations where functional mismatch is about the behavior of a single method. In other case, clients expect an operation in a required interface to call a function. However, a component services two operations for supporting the function. Finally, the degrees of match such as perfect match (FULL), partial match (PARTIAL), and no match (NONE) is identified.

As shown in Table 2, the partial matching descriptions define features satisfied by candidate components and write features not satisfied by candidate components. These descriptions are derived from 2a and 2b activities. We describe partial match problems that include related use case name, requirement specifications, provided interface and functionality of candidate components, and mismatch problems.

4.3 Phase 3. Adaptation Design

The aim of this phase is to obtain adaptation techniques and strategies for adapting candidate components that are partially matched components.

Activity 3a. Select Mechanisms for Adaptation

This activity determines the method such as internal or external adaptation to adapt the candidate components. Internal adaptation uses a customization mechanism [10] in each component. If the partial matching problems are not resolved by the customization method, external adaptation is used.

External adaptation uses smart connectors that fill the gap between candidate components and the specification of components required [1]. By using smart connectors, partially matched components can be reused. The input artifacts are partial match problems, category of partial match and candidate components that have partial matching problems. The output is the adaptation methods to adaptation the candidate components.

We define the taxonomy of partial matching problems as functional, workflow, attribute, data value range, persistency, and interface mismatch problems in Table 1. Internal adaptation can adapt functional, workflow, attribute, data value range, persistency, and interface mismatch problems. External adaptation can adapt functional, workflow, data, and interface mismatch problems.

If the problems can be solved by both internal and external adaptation, then we prefer internal adaptation because internal adaptation is simpler than external adaptation. Internal adaptation has better performance. If functional partial matching problems are fully supported by internal adaptation such as with a customization mechanism, then internal adaptation determines whether or not external adaptation supported the problems. If the mismatch is partially supported by internal adaptation, then external adaptation adapts the partial problem that is not covered by the customization mechanism. The problems are supported by combination techniques.

If workflow partial matching problems are partially supported by internal adaptation, external adaptation is decided. We do not recommend combination method to resolve workflow and persistency partial matching categories. If the

mismatches are adapted by internal and external adaptation, the complexity and side effects are increased. Table 3 is algorithms for determining the adaptation technique. If the partial matching problems can not be solved by the internal and external adaptations, then the candidate component is rejected or the requirement specification of the target system will be modified.

Table 3. Algorithm for Determining Adaptation

Category	Degree of Adapting using Customize	Degree of Adapting using Connector	Identified Method
Attribute	Full	—	Internal
	None	—	Reject Comp.
Functionality	Full	—	Internal
	Partial	Full	Combination
		Partial or None	Reject Comp.
	None	Full	External
		Partial or None	Reject Comp
Interface	Full	—	Internal
	Partial	Full	Combination
		Partial, None	Reject Comp.
	None	Full	External
		Partial, None	Reject Comp
Workflow	Full	—	Internal
	Partial	Full	External
		Partial or None	Reject Comp.
	None	Full	External
		Partial, None	Reject Comp.
Persistency	Full	—	Internal
	Partial, None	—	Reject Comp.

Activity 3b. Refine Adaptation Mechanisms

The purpose of this activity is to describe how to adapt partial matching problems. This activity describes strategies to reuse components that have partial matching problems. The requirement set for adaptation through internal and external adaptations is defined. Table 4 shows adaptation for each mismatched situation.

If the selected adaptation technique is internal adaptation, the customize mechanism is described to adapt partial matching problems. The description consists of related components information, partial match problems, customize interface, feature of compatible variants, pre-condition and post-condition. The candidate components are adapted by these descriptions in the activity 4.1 of the next phases.

If the selected adaptation technique is external adaptation, the description focuses on the requirement specification of the smart connector. The requirement specification consists of related components information, partial matching problems, connector types, connector ports, connector roles, pre-conditions, and post-conditions. The connector types are *interface adapter, value range transformer, functional transformer,* and *workflow handler* [1]. Partial matching problems between

components are resolved by smart connectors that are implemented according to the requirement specification of the smart connector in activity 4.2 of the next phases.

Table 4. Adaptations Required for Partial Matching

Partial Match	Types of Partial Match	Adaptation Required
Attribute	Some attributes required	Appending additional attributes
	Some extra attributes presented	Disabling extra attribute
	Data type of attribute varies	Transforming attribute type
Func-tionality	Some functionality required	Appending additional behavior
	Some extra functionality presented	Disabling extra behavior
Interface	Name of operation varies	Adapting operation names
	Type of parameter or a return type varies	Transforming types
	Value range of a parameter or a return type varies	Transforming value ranges
	Order of parameters for an operation varies.	Rearranging order of parameters
Workflow	Operation of workflow type required	Revising operation of workflow
	Order of invocations in a workflow varies	Rearranging order of workflows
Persistency	Persistent database model employed varies	Adapting database models
	Schema for relational tables varies	Adapting database schemas

4.4 Phase 4. Component Adaptation

The last phase is to realize adaptation techniques such as customizing candidate components and gluing smart connectors.

Activity 4a. Apply Adaptation Mechanisms

The purpose of this activity is to realize adaptation techniques such as customizing candidate components and creating smart connectors.

If components should be customized, the candidate components are adapted using a customization mechanism [10] through the plan of phase 3. Candidate components that need internal adaptation or a combination of internal and external adaptation are applied. Input artifacts are determined by the adaptation technique, adaptation descriptions, and candidate components with the manual for customizing. An output artifact consists of adapted components.

If component should be needed smart connectors, the candidate components are adapted by *smart connector* [1]. Smart connectors resolve candidate components that need external adaptation or combination of internal and external adaptations. Input artifacts are determined by the adapting technique, requirements specifications of smart connectors, and specifications of provided and required interface of candidate components. The Output artifact is a design and implementation of the smart connectors.

Activity 4b. Validate Adapted Components

Adapted components from phase 3 are validated by this activity. Specifications of desired components are compared with adapted components. Test cases should be written in enough detail that they could be given to a new team member who would be able to quickly start to carry out the tests and find defects. Candidate components are compared with its specifications.

This activity also resolves conflicts between internal and external adaptations. It identifies dependency between mismatches and rejects candidate components. During the adapting partial matching, some adapted mismatches affect other components. If functions or interfaces that depend on other components are adapted, then unexpected side effects can be generated. Therefore, these side effects are considered and resolved.

For example, if adaptation techniques adapt the workflows to resolve partial matching problems but side effects are not resolved, the candidate component should be rejected. Note that side effects such as state changes and database updates caused by invoking new function, data value range, workflow and persistency must be carefully examined to maintain the integrity of components.

It validates the applicability of components. The adapted component meets the needs and expectations of the customer. The adapted components are tested with test cases for both functional and non-functional requirements such as performance issues. If the components do not satisfy the requirements, the components should be revised or rejected.

5 Conclusion

In CBD, if the behavior provided by components does not exactly match the specification of the desired components, they have to be adapted for the specific requirements of each application. In this paper, we first identified and organized commonly occurring forms of partial matching into taxonomy of partial matching. Then, we proposed a process and instructions for component adaptation as a way to resolve the problem of *partial matching*.

For internal adaptation, the partial matches between a candidate component and the specification of the desired component can be resolved through pre-defined customization interface or the mechanism of candidate components using *component customization*. If the component does not provide adequate customizability for the partial match, then the component has to be adapted through an external adaptation mechanism such as *smart connectors*. By using connectors, partially matched components become reusable in application development without sacrificing the component consumer's requirement.

The process proposed in this paper has 4 phases; acquiring candidate components, identifying gaps between candidate components and desired components, defining the adaptation technique, and applying the selected adaptation. We also provided

instructions and an internal and external mechanism for these phases. By using the proposed framework, we believe that the reusability, applicability, customizability and maintainability of black-box components can be greatly increased.

References

[1] Min, H., Choi, S., and Kim, S., "Using Smart Connectors to Resolve Partial Matching Problems in COTS Component Acquisition," *Proceedings of 7th International Symposium on Component Based Software Engineering (CBSE 2004), LNCS 3054*, pp. 40-47, 2004.

[2] Ravichandran, T., and Rothenberger, M., "Software Reuse Strategies and Component Markets," *Communications of the ACM, Volume 46, Issue 8*, pp. 109-114, 2003.

[3] Kim, S., Her, J., and Chang, S., "A Theoretical Foundation of Variability in Component-based Development," *Journal of Information and Software Technology*, *Volume 47, Issue 10,* pp. 663-673, 2005.

[4] D'Souza D., and Wills A., *Objects, Components, and Frameworks with UML*, Addison Wesley, 1999.

[5] Keepence, B., and Mannion, M., "Using patterns to model variability in product families," IEEE Software, Vol. 16, Issue. 4, July-Aug., 1999.

[6] Anastasopoulos, M., and Gacek, C., "Implementing Product Line Variabilities," *Proceedings of the 2001 symposium on Software reusability: putting software reuse in context*, Toronto, Canada, May 2001.

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

[8] Gomaa, H., *Desiging Software Product Lines with UML*, Addison-Wesley, pp. 6-7, 2004.

[9] Angrawal, S., Narasayya, V., and Yang, B., "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design," SIGMOD 2004, pp359-370, 2004.

[10] Kim, S., Min, H., and Rhew, S., "Variability Design and Customization Mechanisms for COTS Components," *Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2005)*, *LNCS 3480*, pp. 57-66, 2005.

[11] OMG, Unified Modeling Language: Superstructure Version 2.0, ptc/03-08-02, 2003.