Behavioral Mismatch of Component Adaptation in Component-Based Software Engineering

Poonam Kumari1, Ishdeep Singla2
1M.Tech Student 2Assistant Professor
1,2Chandigarh University Gharuan

Abstract The main goal of Component-Based Software Engineering is Reuse of existing software components. Adaptation is a crucial problem in the component-based software engineering. Components that assembled or reused sometimes cannot perfectly fit one another because of the incompatibility issues between them. Behavioral mismatches are one of the adaptation problems. This paper focus on detect the behavior mismatches and how to create adapter to overcome incompatibility problem. The model based approach and Symbolic Transition Systems that used in representing component interface are used to used to detect the mismatches and overcome them.

Keywords -- Software components, interfaces, Behavioral mismatch and software adaptation

I. INTRODUCTION

Software systems can be developed by selecting commercial off-the-shelf components and then assembling them with well-defined software architecture. Component-based software engineering (CBSE) combines elements of software architecture, modular software design, software verification, configuration and deployment [1]. Component-based software engineering (CBSE) is an approach to software development that relies on software reuse. Reuse of software requires a certain degree of adaptation, particularly in the presence of legacy code. Mismatch problem is often faced by the developers during assembling and reusing of existing components [7]. Integration among components needs adaptation to solve this problem.

Component Adaptation: Component adaptation is widely recognized to be one of the critical problems in Component-Based Software Engineering [1]. The capability of adapting off-the-shelf software components to work properly within larger applications is a must for the development of a true component marketplace and for component deployment in general. The component adaptation process helps to guarantee that software components are able to interact with each other's successfully.

A component must be reusable from its interface, which in fact equivalent to its full technical specification. There are several levels of mismatches of component interface: technical level, signature level, behavioral level, semantic level and service level. The behavioral mismatch can be caused by not correspond message names, incompatible ordering of messages in two or more components, or by some messages in one component that have no match with several messages in another component. The behavioral level mismatches and how to ensure web services compatible at the behavioral level is an important issue for web services integration [3]. Deadlock is a typical ‘behavioral mismatch with reordering’ in which ordering of messages cause the mismatch. There are two classification of adaptation of existing reuse unit that are white-box and black-box. White-box adaptation needs to understand internal component while the black-box adaptation needs to understand just interface of subject of reuse.

II. COMPONENT ADAPTATION TECHNIQUES

Component-based software engineering (CBSE) intends to build applications by putting together the reusable components. The software construction based on components as exists within the software engineering community. The process of changing the component for use in a particular application is often referred to as component adaptation. The software engineering has developed number of techniques for adapting components [5]. These component adaptation techniques can be categorized into white-box and black-box adaptation techniques.

– White-box techniques require the software engineer to adapt a reused component either by changing its internal specification or by overriding and excluding parts of the internal specifications

– Black-box techniques reuse the component as it is, but adapt at the interface of the component. Black-box adaptation only requires the software engineer to understand the interface of the component, not the internals specifications.

III. COMPONENT INTERFACES

Component Interfaces are PeopleSoft's way of exposing the business logic developed into components for consumption by other areas of the system [1]. Component interfaces are part of PeopleSoft's Integration Broker technology and an attempt to introduce SOA into the product. They tend to work quite well but can be slow for large amounts of data processing. Component interfaces are used both signature and behavioral interface.

– Signature interfaces usually correspond in component-based frameworks (e.g., CCM, .NET, or J2EE) to operation profiles described using an IDL, i.e., operation names associated with argument and return types relative to the data being exchanged when the operation is called. Industrial component models, by using Interface Description Languages (IDLs), are able to solve the most of technical interaction problems, but they fail to address mismatch at the higher levels.

– Since we focus on the behavioral level, the elements relative to data exchange in the signature interfaces. This means that a signature is taken as a disjoint set of provided and required operation names. Such abstractions from data exchange are often used in software engineering, e.g., to check interface compatibility or to perform component verification. Many approaches have been presented for extending component interfaces with protocols thus resulting in what we call Behavioral IDLs (BIDLs). This interoperability level is
essential because, even if components match from a signature point of view, their combination can lead to incorrect behaviors or deadlock situations if the designer is not aware of their execution flows and does not take them into account while building the full system.

IV. BEHAVIORAL MISMATCH

Several approaches and techniques developed to overcome behavioral mismatch problems. All of them aim to build one common solution called adapter, but they are differ in the used algorithms and techniques. Mismatch situations between component interfaces may be caused by message names that do not correspond, by an ordering of messages which is not compatible in two or more components, or by some messages in one component that have no counterpart or match with several messages in another component (one-to-zero, one-to-many, or many-to-one correspondences). All of these cases of behavioral mismatch can be used the contract notation and the adaptation algorithms [4]. The deadlock is used to check the mismatch of component behavior because deadlock is most commonly accepted notation. The behavior of adapter can be calculated automatically from the adapted components and the adaptor specification. The adapter can make the correctness of the component interaction with data exchange, and realize the reordering of message. A model-based adaptation approach proposed to focus on mismatch between behavioral interfaces of two components[10]. It is automatically generate adaptor based on Labeled Transition Systems (LTSs) and by cutting off the behaviors causing mismatches.

A. Model-based adaptation approach

A model-based adaptation approach focusing on mismatches appearing at the behavioral level. The model-based adaptation approach are proposed two algorithms: the first based on synchronous products and the second based on Petri net encodings [8]. The adapter implements both to use them with a user-friendly graphical interface.

![Image](image.png)

Fig. 1: Model-based adaptation approach.

Model-based adaptation approach for software adaptation proposed behavioral interfaces. These are represented by means of Labeled Transition Systems (LTSs). The synchronous product of several component Labeled Transition Systems (LTSs) results in new Labeled Transition Systems (LTSs), which contains all of the possible interactions between the involved components. They depend on synchronous vectors, which denote communication between several components.

A novel formal model for web service interfaces, which used to reach to and use the services. They give the condition for detecting behavioral mismatches among multiple web services, which relies on an abstract notation based on Labeled Transition Systems (LTS)[3]. To support this method, an algorithm automatically builds the synchronous product for a set of web service behavioral interfaces. The safety and effective approach of service behavior adaptation presented a coordinator, which used for interaction of web services and solution for behavior mismatch. The author presented their approach with the calculation model and the framework of service behavior adaptation. The behavior protocol of adaptor –according to the specification of adaptor in which each element called as synchronous vector [3]. Therefore, the adaptor will interact with services according to its behavior protocol. The service behavior specified using the Symbolic Transition Systems (STS).

To calculate the behavior protocol of adaptor, there is a need to calculate the synchronous product of the Symbolic Transition Systems (STS) specification of components and the specification of adaptor. The synchronous vectors indicate communication between several components, where each event appearing in one vector is executed by one component and the overall result corresponds to synchronization between all of the involved components [4].

To overcome the behavior mismatch with the help of Component Interaction Adaptation Model (CIAM) [3]. Before proposing this model the Qi Huacheng, Rong Mei and Zhang Guang quan build component behavior ontology (CBO). This includes component entity, component behavior and component behavior properties[9]. The Component Interaction Adaptation Model (CIAM) is based on the component behavior ontology (CBO). This model describe in two phases to overcome the behavior mismatch.

The first phase is detecting the behavioral mismatch, by dividing different components groups and detect interaction behavior deadlock.

The second phase is adapter, this phase define behavior rule, construct adapter specification by behavior rule and get an adapter to solve the deadlock.

Researchers calculate synchronous product of Symbolic Transition Systems (STS), which are the abstract specification of component behavior to detect automatically deadlock mismatch.

V. CONCLUSION

Adaptation is a critical issue when building new applications by reusing existing services and components. The integration between components usually presents different mismatches according to multiple levels. Behavioral mismatches are one of the adaptation problems. Researchers have been present different solutions for that problem like adapter and some other techniques. The solution suggests focusing on data type through representing component interface by Symbolic Transition Systems (STS) and through calculating synchronous vector.

In the future, the authors want to implementing adapter tool and looking for improve the proposed tool to be able to solve other mismatches either in the behavioral level or in the other levels.
REFERENCES


