Verifying the compatibility of components’ ports upon specification

Nguyen Hoang Ha, Tran Thi Mai Thuong, Truong Ninh Thuan, Nguyen Viet Ha
College of Technology, Vietnam National University, Hanoi
144 Xuan Thuy, Cau Giay, Hanoi
Email: thuantn@vnu.edu.vn, hanv@vnu.edu.vn

Abstract. We propose in this paper an approach for verifying the compatibility between components’ ports upon specification. In component software development, ports are the points of interaction between components. The connection between ports must satisfy some constraints. We determine these constraints and propose to use the B method and its support tools for verifying the compatibility between ports in a component model.

1 Introduction

We are seeing an enormous expansion in the use of software in business, industry, administration, research, and even in everyday life. Features based on software functionality, rather than other system characteristics, are becoming the most important factor in a competitive market. This trend increases the demands on software products such as enhanced usability, robustness, reliability, flexibility, adaptability, and simpler installation and deployment. As these demands are growing stronger, the complexity of processes that software manages is increasing along with the demand for the integration of processes from different areas. As a consequence, software programs are becoming increasingly large and complex. The appearance of component based software engineering adapts this challenge of the software development, it proposes an easy and efficient method for developing large software.

Component-based Software Engineering (CBSE) is concerned with the development of software intensive systems from reusable parts (components), the development of such reusable parts, and with the maintenance and improvement of systems by means of component replacement and customization.

Main feature of the CBSE is to allow the construction of an application using independently developed software components, leading to reduce development costs and improved software quality. In this process it is essential to ensure that individual components can in fact interoperate together in the system. However the components do not interact seamlessly. Problems could arise in the system if there are mismatches and inadequacies of connect points between components. In the development of CBSE, a component can be considered as a black box, it interacts with the environment through its interfaces. The interface of a component, that is, its external view, is described as a set of ports. The ports are the points of interaction between a component and the environment.
In this paper, we propose an approach to verify the compatibility between components through specifications of their ports. We concentrate on the CORBA Component Model (CCM) ports. Firstly, specification of components is described by XML. We then determine the conditions such that ports can be connected. From the XML description and these constraints, we finally build a B abstract machine which can be used to check the consistency of connected ports in the model.

The B method [7] is used to verify the compatibility between ports. Because, the B notations are based on set theory, generalised substitutions and first order logic, these are easily to describe ports and their relation. In addition, the proof obligations for B specifications are generated automatically by support tools like AtelierB [18], B-Toolkit [16] and B4free [9]. Checking proof obligations with B support tools is automatically performed.

In the following, we present an overview of component based software engineering. We then describe our method in Section 3 and illustrate it with the case study of the Stock Quoter System. In Section 4, we discuss related work. The paper finishes with some concluding remarks in Section 5.

2 Specification of component approaches

A component has many different parts that must be specified for many different purposes and there is a consequent need for different specification techniques. The description of a component is not easy if it is not clear what a component is. Thus, a well-formulated and clearly understood definition of the component concept is needed.

In order to specify software components, ones usually use Unified Modeling Language (UML) and the Object Constraint Language (OCL), in which a component implements a set of interfaces. Each interface consists of a set of operations with associated pre- and postconditions and invariants.

Many architecture description languages for components have been proposed, sometimes to satisfy the needs of different application domains. There is agreement about what the set of core concepts could be, at least regarding the structural aspect of architectural description. Acme is an architectural description language [13] that incorporates these concepts. Acme is a second-generation ADL that focuses on the definition of a core set of concepts related to the structural aspects of software architecture in order to make it possible to exchange descriptions between independently developed ADLs. Acme is a generic models of the different component technologies currently available in the industry: JavaBeans [1], COM+ [11], CCM [2], .NET [3], and the Open Service Gateway Initiative (OSGI) [4].

In this paper, we focus on the verification between ports of CCM (CORBA Component Models) represented by Acme. The CCM is the most recent and complete component specification from OMG [5]. It has been designed on the basis of the accumulated experience using CORBA service, JavaBeans, and EJB. Like many component models, CMM is not only used by the developer who builds applications by assembling available parts, but also used explicitly by the component designer, assembler, and deployer. The major goal behind the CCM specification is to provide a solution to the complexity reached by CORBA
CCM simply defines the concept of connection as an object reference; thus CCM, like all other industrial component models, does not provide a connector concept. Nevertheless, components are connected by linking facets to receptacles and event sources to event sinks. Connections are binaries and oriented, but the same port can handle multiple connections. Connections can be explicitly described (in the assembly descriptor, an XML file) and established by the CCM framework at initialization.

Components support a variety of surface features through which clients and other elements of an application environment may interact with a component. These surface features are called ports. The component model supports four basic kinds of ports [10] (see Figure 1):

- Facets, which are distinct named interfaces provided by the component for client interaction.
- Receptacles, which are named connection points that describe the component’s ability to use a reference supplied by some external agent.
- Event sources, which are named connection points that emit events of a specified type to one or more interested event consumers, or to an event channel.
- Event sinks, which are named connection points into which events of a specified type may be pushed.
- Attributes, which are named values exposed through accessor and mutator operations. Attributes are primarily intended to be used for component configuration, although they may be used in a variety of other ways.

Basic components are not allowed to offer facets, receptacles, event sources and sinks. They may only offer attributes. Extended components may offer any type of port.
3 Verifying the compatibility between ports

To demonstrate our approach, we use a case study of the Stock Quoter System\(^1\). Figure 2 illustrates the components in stock quoter system example using the CORBA Component Model (CCM). The StockDistributor component monitors a real-time stock database. When the values of particular stocks change, it pushes a CCM eventtype that contains the stock’s name via a CCM event source to the corresponding CCM event sink implemented by one or more StockBroker components. If these components are interested in the stock they can obtain more information about it by invoking a request/response operation via their CCM receptacle on a CCM facet exported by the StockDistributor component.

![Fig. 2. CORBA component interface and its ports](image)

```idl
component StockBroker {
    consumes StockName notifier_in;
    uses StockQuoter quoter_info_in;
};
```

StockBroker contains two ports that correspond to the following two roles it plays.

It’s a subscriber that consumes a StockName eventtype called \texttt{notifier\_in} that’s published by the StockDistributor when the value of a stock changes. As shown in Figure 2, the \texttt{notifier\_in} event sink will be connected to the StockDistributor’s \texttt{notifier\_out} event source by the standard CCM deployment and configuration tools when the application is launched.

It uses the StockQuoter interface provided by the StockDistributor component, which reports additional information about a stock, such as the high, low, and most recent trading values of the stock during the day. The dependency of StockBroker on StockQuoter is indicated explicitly in IDL 3.x via the \texttt{quoter\_info\_in} receptacle, which will be connected to StockDistributor’s \texttt{quoter\_info\_out} facade by the deployment and configuration tools when the application is launched.

We now present the implementation of the StockDistributor component, whose ports are shown here:

\(^1\) [http://www.ddj.com/cpp/184403889](http://www.ddj.com/cpp/184403889)
component StockDistributor supports Trigger {
    publishes StockName notifier_out;
    provides StockQuoter quoter_info_out;
    attribute long notification_rate;
};

It publishes a StockName eventtype called *notifier_out* that is pushed to the StockBroker subscriber components when a stock value changes. In addition, it defines a StockQuoter facet called *quoter_info_out*, which defines a factory operation that returns object references that StockBroker components can use to obtain more information about a particular stock. Finally, this component defines the *notification_rate* attribute, which system administrator applications can use to control the rate at which the StockDistributor component checks the stock quote database and pushes changes to StockBroker subscribers.

We now consider the verification of compatibility between ports when component specification in this system are connected.

Recall that information from a component specification can be described by XML. XML (Extensible Markup Language) [6] is a simple, very flexible text format derived from SGML. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. XML can use also to define metamodel or metadata of a system specification. With a XML document described valid CORBA system, it can provide an easy way to extract information about components and its ports for the verification purpose.

The ADL specification of the Stock Quoter System presented in Figure 2 can be described by XML as the following.

```xml
<connections>
    ...<usesport>
        <usesidentifier>quoter_info_in</usesidentifier>
        <componentinstantiationref idref="StockBroker"/>
    </usesport>
    <providesport>
        <providesidentifier>quoter_info_out</providesidentifier>
        <componentinstantiationref idref="StockDistributor"/>
    </providesport>
</connectinterface>
</connectevent>
```
<componentinstantiationref idref="StockDistributor"/>
</publishesport>
</connectevent>
</connections>

Note that, in a CCM specification, if a receptacle’s uses declaration does not include the optional multiple keyword, then only a single connection to the receptacle may exist at a given time. If a receptacle’s uses declaration includes the optional multiple keyword, then multiple connections to the receptacle may exist simultaneously.

There are two categories of event sources, emitters and publishers. Both are implemented using event channels supplied by the container. An emitter can be connected to at most one proxy provider by the container. A publisher can be connected through the channel to an arbitrary number of consumers, who are said to subscribe to the publisher event source. A component may exhibit zero or more emitters and publishers.

A publisher event source has the following characteristics [2]:

– The equivalent operations for publishers allow multiple subscribers (i.e., consumers) to connect to the same source simultaneously.
– Subscriptions to a publisher are delegated to an event channel supplied by the container at run time. The component is guaranteed to be the only source publishing to that event channel.

An emitter event source has the following characteristics [2]:

– The equivalent operations for emitters allow only one consumer to be connected to the emitter at a time.
– The events pushed from an emitter are delegated to an event channel supplied by the container at run time. Other event sources, however, may use the same channel.

As a consequence, CCM components can be connected if their ports satisfy conditions:

– Facet can connect only to receptacles (provides port connect only to uses port)
– Event source can connect only to event sinks (We can say that publishes and emits ports can connect only to consumes ports)
– Each provides port (facet) can connect to many uses ports (receptacles), each publishes port can connect to many consumes ports but not on the contrary.
– Each emits port connect only to one consumes port.
– With two connected ports, type of provided ports (facets, event sources) is a subtype of the one of required ports (receptacles, event sinks).

In order to verify these conditions for connecting ports in a specification, we propose to use the B method [7]. The B machine that we build to verify the correctness of the ADL Acme specification is called the ConnectionCheck. From
the XML description, we can get all ports and type of port (uses port, provides port, consumes ports...) in the specification. They are presented in the SETS clause of the machine.

We declare the variable $\text{connection}$ to get all connections in the schema. The connection have to satisfy all conditions described in the above. These constraints can be formally described in the INварIANTS clause as the following:

\[
\text{connection} \in \text{USESSPORT} \rightarrow \text{PROVIDESPORT} \lor \\
\text{connection} \in \text{CONSUMESPORT} \rightarrow \text{PUBLISHESPORT} \lor \\
\text{connection} \in \text{CONSUMESPORT} \rightarrow \text{EMITSport}
\]

In these constraints, type of the $\text{connection}$ variable defines the type of a possible connection in the specification. We use the partial function ($\rightarrow$) to denote the relation between the domain and the range of the connection between uses port and provides ports; consumes port and publishes port. It means that, one element of the domain cannot connect to have more than one element of the range and one element of the range can connect to many elements of the domain. We use the partial bijection ($\rightarrow$) to denote the relation between consumes port and emits port. It means that each element of the domain can connect only to one element of the range.

In the OPERATIONS clause of the machine, we define operations for extracting all connections in the CCM specification. The machine presented in Figure 3 illustrates the B notations of the verification purpose for the case study of the Stock Quoter System in Figure 2. It is to be noted that, all information in this abstract machine can be extracted from the XML description hence it can be built automatically.

4 Related work

Several proposals for verifying the interoperability between components have been made. The paper [12] present a tool called Cadena, an integrated environment for building and modeling CCM systems. Cadena provides facilities for defining component types using CCM IDL, specifying dependency information and transition system semantics for these types, assembling systems from CCM components, visualizing various dependence relationships between components, specifying and verifying correctness properties of models of CCM systems derived from CCM IDL, component assembly information, and Cadena specifications, and producing CORBA stubs and skeletons implemented in Java.

As a point of comparison, this paper generated a DSpin model for the scenario that check the number of timeouts issued in a system execution.

Zaremski and Wing [19] propose an approach to compare two software components. They determine whether one required component can be substituted by another one. They use formal specifications to model the behavior of components and exploit the Larch prover to verify the specification matching of components. In [14, 15], protocols are specified using a temporal logic based approach, which leads to a rich specification for component interfaces. Henzinger and Alfaro [8] propose an approach allowing the verification of interfaces interoperability based
MACHINE ConnectionCheck

SETS
USESSPORT = \{ quoter\_info\_in \};
PROVIDESPORT = \{ quoter\_info\_out \};
CONSUMESPORT = \{ notifier\_in \};
PUBLISHESPORT = \{ notifier\_out \};
EMITSORTS;

VARIABLES
connection

INVARIANTS
connection ∈ USESPORT → PROVIDESPORT ∨
connection ∈ CONSUMESPORT → PUBLISHESPORT ∨
connection ∈ CONSUMESPORT → EMITSORT

INITIALISATION
connection := ∅

OPERATIONS
getConnectionU_\_P =
PRE
connection ∈ USESPORT → PROVIDESPORT THEN
  connection := \{ notifier\_in → notifier\_out \}
END;

getConnectionC_\_P =
PRE
  connection ∈ CONSUMESPORT → PUBLISHESPORT THEN
  connection := \{ quoter\_info\_in → quoter\_info\_out \}
END;

getConnectionE_\_C =
PRE
  connection ∈ CONSUMESPORT → EMITSORT THEN
  connection := ∅
END
END

Fig. 3. B abstract machine for verifying compatibility between component ports
on automata and game theories: this approach is well suited for checking the interface compatibility at the protocol level. The paper [17] proposes the Port State Machine (PoSM) to model the communication on a Port. Building on their experience with behavior protocols, they model an operation call as two atomic events request and response, permitting PoSM to capture the interleaving and nesting of operation calls on provided and required interfaces of the Port. The trace semantics of PoSM yields a regular language. They apply the compliance relation of behavior protocols to PoSMs, allowing to reason on behavior compliance of components in software architectures.

Our work focuses on the verification of interoperability of specification of components through their ports. We determine the conditions for the connection between ports and use the B method for verifying their compatibility.

5 Conclusion

Increasingly, complex software systems are being constructed as compositions of reusable software components. One critical issue for such constructions is the definition and verification which kinds of component ports can be compatible. If the connection between components is correct, it will permit the various software components to work together properly. However, it is not always possible to found that which components can be combined to satisfy the requirements of the system, and at present it is time-consuming and difficult. Definition of constraints on component ports as well as verification of correctness of the compatibility between components is needed.

In this paper, we defined constraints on types of port and basing on these we can know which components can connect together properly if their ports satisfy requirements which we given. A part from this, we have proposed an approach to verify the compatibility between components by building B machine that is used to verify the correctness of the ADL Acme specification. And to demonstrate our approach, we used a case study of the Stock Quoter System. As a result, from the XML description, we could get all ports and type of port (uses port, provides port, consumes ports...) in the specification and then verified the correctness of the compatibility between components whose types of port satisfy constraints described above.

Like this, at the first degree, we know only types of port (facet, receptacle, event source, event sink) and possibly do not know behavior of ports, we have just verified the compatibility between types of port connecting between components. However, we haven’t verified the condition that with two connected ports, type of provided ports (facets, event sources) is a subtype of the one of required ports (receptacles, event sinks). In the future work, we will carry out to verify the rest condition and simultaneously in the latter degree, we will check the composition between behaviors of ports which can be expressed by Port State Machine (PoSM) [17] when connection between types of port is correct.

Acknowledgments. This work is partly supported by the research project No. QC.07.04 granted by Vietnam National University, Hanoi.
References