VNU Journal of Science, Natural Sciences and Technology 24 (2008) 92-102

Checking the conformability in CORBA component model specifications

Tran Thi Mai Thuong*, Vo Van Thanh, Truong Ninh Thuan

College of Technology, VNU, 144 Xuan Thuy Road, Cau Giay District, Hanoi, Vietnam

Received 31 October 2007

Abstract. We proposed in this paper an approach for checking the conformability in CORBA component model specifications. In software engineering, it is demonstrated that discovering bugs in earlier phases is much more economical than later phases. We focused thus on verifying components by their ports specification. In order to do this, firstly we determined constraints on kinds of port as well as on types of port which the connection between ports must satisfy, and then formalized them to be able to prove automatically using formal prover tools. Here, we proposed to use the B method for verifying components in a CCM specification.

1. Introduction

The enormous expansion in the use of software in every field of life make demands on installing and developing reusable, robust, reliable, flexible, adaptive software systems much accelerating. 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 (CBSE) adapts this challenge of the software development; it proposes an easy and efficient method for developing large software.

In this approach, the architecture of a system is described as a collection of components (reusable parts) along with the interactions among those via their ports. The main feature of 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 connected points between components. It is important to verify the correctness of component composition. In order to do it, there are many approaches appeared to verify the compatibility between components by interfaces [1,2], behaviour specification [3], models [4]...

As we know, however, CBSE is also an approach to develop software systems, hence

* Corresponding author. E-mail: thuongttm@vnu.edu.vn

discovering bugs in the earlier phases will reduce much time and effort in building software systems, especially large systems. So, in this paper, we propose an approach to verify the conformability between components through specifications of their ports. This is a buffer step before verifying behaviour specifications of components, because it will remove many unneccesary cases which are inputs for checking behaviours.

Here, we use the CORBA Component Model (CCM) ports. Firstly, CCM specification of components is described by XML. We then determine the conditions such that ports can be connectable. From the XML desciption 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 [5] 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 [6], B-Toolkit [7] and B4free [8]. Checking proof obligations with B support tools is automatically perfomed.

In the following, we present an overview of components specifying approaches. 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 software components

Specification of software components is one of the most important research challenges in component-based software engineering. It represents the first step towards true component reuse as the component specification gives all necessary information to the component user on how/why the component can be (re) used.

A component is considered to be a black box. Hence, interfaces are the only access points to the component and the specification of the component comes down to the specification of the component interfaces.

Specification of the component interfaces in the current component-based systems is done by two levels:

- On the first level, syntactical level, there are some specification models such as JavaBeans [9], COM+ [10], CCM [11], .NET [12], and the Open Service Gateway Initiative (OSGI) [13]. At this level, The component specification consists of specification of provided and required interfaces. Provided interfaces are the one that contain operations that a component provides to other components or to the component user, while required interfaces are the one that contain operations used by the component.
- On the second level, semantic specification, there are two representatives: 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, as well as component state and invariants. Preconditions are assertions that the component assumes to be fulfilled before an operation is invoked, while postconditions are assertions that the component guarantees will hold just after the operation has been invoked. An invariant is a predicate over the interface's state that will always hold.

In this paper, we focus on verifying the conformability between components by ports in CCM (CORBA Component Models). The CCM

is the most recent and complete component specification from OMG [14]. It has been designed on the basis of the accumulated experience using CORBA service, JavaBeans, and EJB. The major goal behind the CCM specification is to provide solution to the complexity reached by CORBA and its services. One of the advantages of CCM is its effort to integrate many of the facets involved in software engineering. As a consequence, a software application is described in different formalisms along two dimensions: the time dimension (the life cycle, from design to deployment) and the abstract dimension (from abstractions to implementation). Altogether, this makes a rather complex specification.

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 [15] (see Figure 1):

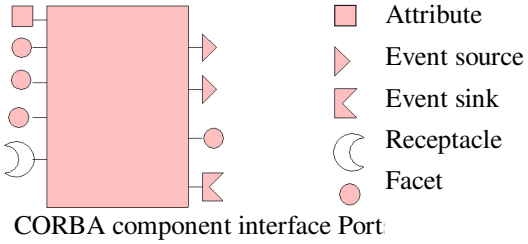

Fig. 1. CORBA component interface and its ports.

- 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.

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. Case study: Stock Quoter System

To demonstrate our approach, we use a case study of the Stock Quoter System[1] with two components connected by their ports. However, our approach will work with more complex systems in which there are many connected components. According to this approach, we

[1] http://www.ddj.com/cpp/184403889

firstly transform CCM specification of components into XML format. We then express XML description and constraints which we defined above as inputs of B abstract machine. Finally, we use an automatic proof tool to check the consistency of connected ports in the model with B abstract machine.

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.

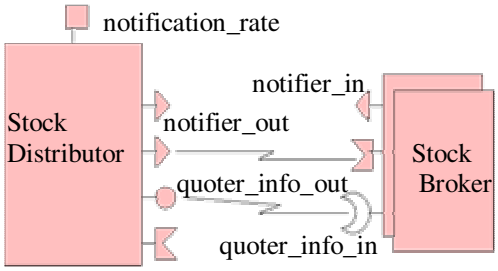

Fig. 2. CORBA component interface and its ports.

```
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 *notifier_in* that's published by the StockDistributor when the value of a stock changes. As shown in Figure 2, the *notifier_in* event sink will be connected to the StockDistributor's *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 *quoter_info_in* receptacle, which will be connected to StockDistributor's *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:

```
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 conformability between components when we have information describing the connection between ports of components from their CCM specification in this system.

Recall that information in component specification can be described by XML. XML (Extensible Markup Language) [16] 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 also use 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.

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 [11]:

- 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 [11]:

- 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:

PD1. Facet can connect only to receptacles (provides port connect only to uses port)

PD2. Event source can connect only to event sinks (We can say that publishes and emits ports can connect only to consumes ports)

PD3. 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.

PD4. Each emits port connect only to one consumes port.

PD5. With two connected ports, type of provided ports (facets, event sources) is a subtype of the one of required ports (receptacles, event sinks).

3.1. Checking types of port in connections

Each component is described in a component based model with two phases. The first one is the type, represents the functional interface of the component, what is visible by other components. The second one is the implementation, describes the contents of the component.

The aim of separation of a component description into a type and an implementation is the point of view of the component. Describing the type means specifying the component interface, expressing how it is seen from an external point of view. On the other hand, the implementation represents the interior. In practice, the description of the type and the implementation may be done by different persons, each of them dealing with one step in the refinement of the architecture description, from the top level to the detail level.

An inheritance mechanism exists to describe the components. It may be used for both the types and the implementations. This mechanism is useful to refine a description by overwritting an already existing component.

Restrictions exist, which must be respected. Thus, a component type may inherit from another component type of the same category. In the same way, a component implementation may inherit from another component implementation of the same category.

The final condition of the compatibility between ports (PD5) states that, type of provided ports is a subtype of the one of required ports. A verification shound be considered to ensure the conformity between the types and directions of the connected ports.

In order to verify conditions for connecting ports in a CCM specification, we propose to use the B method [5].

From the inheritance relationship between types of ports, we create a simple B abstract machine called Types machine (Figure 3). In this machine, if an interface TYPE1 inherits from an interface TYPE2, we define TYPE1 is subtype of the TYPE2 (TYPE1 $\subseteq$ TYPE2).

MACHINE Types

CONSTANTS

TYPE1, TYPE2, TYPE3...

PROPERTIES

TYPE1 $\subseteq$ *TYPE2; TYPE3* $\subseteq$ *TYPE2...*

END

Fig. 3. Types abstract machine.

From the Types machine, if we want to check the consistency of the type between two ports in a connection, we have to get the type of required port (TYPE1) and the type of provided port (TYPE2). Each time we get a connection, we have to give a fragment specification as the following into the B specification, according to the definition of subtype:

ANY conn WHERE

conn ∈ TYPE2

THEN

conn : ∈ TYPE1

END

The B prover will check if TYPE2 is a subtype of TYPE1 from this specification.

3.2. Checking kinds of port in connections

The B machine that we build to verify the correctness of the ADL Acme specification [17] is called the ConnectionCheck. From the XML description, we can get all ports and the kind of port (uses port, provides port, consumes ports...) in the specification. They are presented in the SETS clause of the machine.

We declare the variables *connectionU_P* to contain and check the connection between uses ports and provides ports, *connectionC_P* to contain the connection between consumes ports and publishes ports, *connectionC_E* to contain the connection between consumes ports and emits ports. These variables have to satisfy four conditions (PD1, PD2, PD3, PD4) described in the above. These constraints can be formally described in the INVARIANTS clause as the following:

$connectionU\_P \in USESPORT \leftrightarrow PROVIDESPORT \wedge$

$connectionC\_E \in CONSUMESPORT \rightarrow EMITSPORT \wedge$

$connectionC\_P \in CONSUMESPORT \leftrightarrow PUBLISHESPORT$

In these constraints, type of these three variable define the type of a possible connections in the specification.

We use the partial function ($\nrightarrow$) 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 (Figure 4). We use the partial bijection ($\rightarrowtail\!\!\!\!\!\!\rightarrow$) 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.

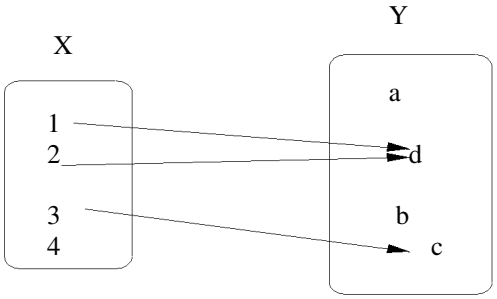

Fig. 4. Relations in a partial function.

In the OPERATIONS clause of the machine, we define operations for extracting all connections in the CCM specification. In these operations, we intergrate the fragment specification of checking types between ports of the connection. The machine presented in Figure 5 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 [4] 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 [18] 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

```
MACHINE ConnectionCheck
SEES Types
SETS
USESPORT = {quoter_info_in};
PROVIDESPORT = {quoter_info_out};
CONSUMESPORT = {notifier_in};
PUBLISHESPORTS = {notifier_out}; EMITSPORTS;
VARIABLES
connectionU_P, connectionC_P, connectionC_E
INVARIANTS
ConnectionU_P ∈
        USESPORT →PROVIDESPORT ∧
ConnectionC_P ∈
        CONSUMESPORT →PUBLISHESPORT ∧
ConnectionC_E ∈
        CONSUMESPORT → EMITSPORT
INITIALISATION
ConnectionU_P := ∅ ||
connectionC_P := ∅ || connectionC_E := ∅
OPERATIONS
GetConnectionU_P =
PRE
ConnectionU_P  USESPORT →PROVIDESPORT
```

```
THEN
ConnectionU_P :=
connectionU_P ∪{notifier_in →notifier_out} ||
ANY conn WHERE /* Check type of ports */
conn  STOCKNAME /* type of provides port */
THEN
conn : ∈  STOCKNAME /* type of uses port */
END
END;
getConnectionC_P =
PRE
connectionC_P ∈
          CONSUMESPORT →PUBLISHESPORT
THEN
connectionC_P := connectionC_P ∪
          {quoter_info_in →quoter_info_out} ||
ANY conn WHERE /* Check type of ports */
conn ∈   STOCKQUOTER /* type of publishes port */
THEN
conn : ∈  STOCKQUOTER /* type of consumes port */
END
END;
getConnectionC_E =
PRE
connectionC_E ∈ CONSUMESPORT → EMITSPORT
THEN
connectionC_E := connectionC_E ∪ ∅…
END
END
```

Fig. 5. B abstract machine for verifying compatibility between component ports.

In [1,2], protocols are specified using a temporal logic based approach, which leads to a rich specification for component interfaces. Henzinger and Alfaro [19] propose an approach allowing the verification of interfaces interoperability based on automata and game theories: this approach is well suited for checking the interface compatibility at the protocol level.

The paper [3] 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

We have presented some aspects of component specifications, outlined our approach of components vefification based on kinds of connectable ports, through proving the correctness of their CCM specification using B method. Concurrently, we also described more detail the transformation from ports' informal connection constraints to formal formats to be able to input into B machine for verifying. We have presented a small but illustrative case study, showing in particular kinds of ports which can be connectable as well as the activity machenism of B machine in proving the soundness of CCM specification.

In previous work, we defined constraints on ports, and thanks to these we can know which components can connect together properly if their ports satisfy requirements which we given. At this degree, we have just only known kinds of port (facet, receptacle, event source, event sink) and only verified constraints on these kinds of port. In this paper, we contributed to verifying connection conditions on types of port and integrating it into kinds of port to assist our approach. This will support us much on verifying the compatability between components by behaviour specification at semantic level.

In the future work, we will carry out to check the composition between behaviors of ports when connection between types of port is correct. Since then, we will build a framework supporting the process of installing, verifying and developing component-based systems. This leaves the opportunity for the designer to use the tool best suited to the problem, and to perform formal analysis on parts of the system that particularly deserve it.

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

References

[1] J. Han, "A comprehensive interface definition framework for software components", In Asia Pacific software engineering conference, *IEEE Computer Society* (1998) 110.

[2] J. Han, Temporal logic based specification of component interaction protocols, In *Proceedings of the Second Workshop on Object Interoperability ECOOP'2000*, Springer-Verlag, (2000) 12.

[3] V. Mencl, "Specifying component behavior with port state machines", Electronic Notes in Theoretical Computer Science, Special issue: *Proceedings of the Workshop on the Compositional Verification of UML Models,* 101C:129, 2004.

[4] J. H. et al. "Cadena: an integrated development, analysis, and verification environment for component-based systems", In *Proceedings of 25th International Conference on Software Engineering,* (2003) 160.

[5] J.R. Abrial, *The B-Book, Assigning Programs to Meanings,* Cambridge University Press, 1996.

[6] Steria, Obligations de preuve: Manuel de ref´erence´. Steria - Technologies de l'information, version 3.0, Available at http://www.atelierb.societe.com.

[7] B.C. Ltd. B-Toolkit User's Manual, Oxford (UK), 1996, Release 3.2.

[8] Clearsy, B4free. Available at http://www.b4free.com, 2004.

[9] Sun Microsystems, JavaBeans 1.01 Specification, http://java.sun.com/beans.

[10] G. Eddon, H. Eddon, Inside COM+ Base Services. Microsoft Press, 2000.

[11] CORBA Component Model Specification, Version 4.0, http://www.omg.org/cgi-bin/doc?ptc/2006-05-01.

[12] Microsoft, .NET, http://www.microsoft.com/net/.

[13] OSGI, OSGI Service Gateway Specification, Release 1.0, http://www.osgi.org.

[14] http://www.omg.org.

[15] I. Crnkovic, M. Larsson. "Building reliable componentbased Software Systems", Artech House, Inc, 2002.

[16] World Wide Web Consortium, XML, http://www.w3c.org/XML/.

[17] Nguyen Hoang Ha, Tran Thi Mai Thuong, Truong Ninh Thuan, Nguyen Viet Ha, "Verifying the compatibility of components' ports upon specification", In *Japan Vietnam Workshop on Software Engineering 2007*, September 2007.

[18] A. M. Zaremski, J. M. Wing, "Specification matching of software components" 6(4) (1997) 333.

[19] L. Alfaro, T.A. Henzinger, "Interface automata", In *9th Annual Symposium on Foundations of Software Engineering*, ACM press, (2001) 109.

ANNEX - XML specification for CMM Stock Quoter System.

```
<connections>

<?xml version="1.0" encoding="UTF-8"?> <Model>

...

<connectinterface> <usesport>

<usesidentifier>quoter_info_in</usesidentifier> <type>StockQuoter</type>
<componentinstantiationref idref="StockBroker" />

</usesport> <providesport>

<providesidentifier>quoter_info_out</providesidentifier>
<type>StockQuoter</type> <componentinstantiationref
idref="StockDistributor" />

</providesport> </connectinterface> <connectevent>

<consumesport> <consumesidentifier>notifier_in</consumesidentifier>
<type>StockName</type> <componentinstantiationref idref="StockBroker" />

</consumesport> <publishesport>

<publishesidentifier>notifier_out</publishesidentifier>
<type>StockName</type> <componentinstantiationref
idref="StockDistributor" />

</publishesport> </connectevent>

</Model>

...

</connections>
```