Ensuring Compatible Interactions within Component-based Software Systems

Jun Han and Ksze Kae Ker
School of Information Technology
Swinburne University of Technology
John Street, Hawthorn, Vic. 3122, Australia
jhan@it.swin.edu.au, kker@it.swin.edu.au

Abstract

The interface definition of a component in a distributed system forms the contract between the component itself and its neighbouring components regarding the use of its services. In general, such a contract should cover the issues of service functionality, usage and quality. The Interface Definition Languages (IDLs) used by commercial middleware standards such as CORBA primarily address the signature issues of such a contract, i.e., the forms and types of component or object services. Nothing is said about other aspects of the contract, including the way in which the component services are to be used. In this paper, we introduce a framework and associated techniques that augment commercial IDLs with interaction protocol specifications and validate component interactions against such protocol specifications at run-time. In effect, the validation becomes a useful tool for testing whether or not the object services are used properly in a distributed system. Our approach has been implemented in the CORBA context, but can be readily applied to other IDL-based object/component systems.

Keywords: Component-based systems, interaction protocols, component interoperability, system validation.

1. Introduction

The use of object and component middleware technologies in building distributed software systems in recent years has gained popularity due to its promise of better reliability, performance, scalability and more importantly, interoperability between heterogeneous systems[1] [26]. Specifically, the success of an industrial middleware standard such as CORBA lies in the fact that its Interface Definition Language (IDL) helps to integrate applications written in different programming languages. Interface definitions define services provided by server objects in a programming-

language neutral manner. Clients written in any language supported by the middleware can communicate with these objects according only to the interface definitions. As such, the interface definitions serve as contracts between the service providers (the server objects) and the service consumers (the client objects).

Nevertheless, current commercial IDLs (including CORBA IDL) have their limitations. These limitations were the main topics at the International Workshops on Object Interoperability held in 1999 [35] and 2000 [15]. At WOI'99, three main levels of interoperability between objects were identified: the signature level (names and signatures of operations), the protocol level (ordering between exchanged messages and blocking conditions), and the semantic level ("meaning" of operations). However, commercial IDLs only address signature issues and nothing on the usage, capabilities and behavior of objects. This implicitly means that incorrect assumptions made about the services of remote objects may lead to incorrect usage, and therefore system failure.

The Rich Interface Definition Language (RIDL) [13] was proposed to address the limitations in commercial IDLs. In addition to the signature-level issues, RIDL also considers service semantics, usage protocols of services, and qualities of service. In particular, the protocols concerning the proper use of an object's services, called *interaction protocols*, specify the service invocation sequences relevant to the object. The explicit definition of the interaction protocols of a server object helps the client objects (or more precisely, their developers) to better understand the server object and avoid problems in using its services. To further increase the confidence about the proper use of object services at run-time, the interactions between the objects in a distributed system can be validated against the defined interaction protocols through testing.

In this paper, we introduce a framework and associated techniques that augment CORBA IDL with interaction protocol definitions and validate object interactions against such protocol definitions at run-time. In effect, the valida-

---

[1]In this paper, we use "component" and "object" interchangeably.

tion becomes a useful tool for testing whether or not the object services are used properly in a distributed CORBA system. The specification of interaction protocols takes the form of temporal constraints, and the run-time validation is fully automatic.

The paper is organized as follows. Section 2 introduces an example to highlight the need for specifying and validating component interactions in CORBA systems. Section 3 presents an overview of specifying component interaction protocols as temporal constraints, in the context of the RIDL. Then, we introduce the framework for validating interaction protocols in CORBA systems in section 4. It involves the use of portable interceptors for tracking component interactions, the FSM-based internal representation of temporal constraints, and the conformance checking of component interactions against the temporal constraints. Section 5 discusses related work before we summarise the key contributions and future directions in section 6.

2. The motivation

In this section we present a scenario that highlights the need for the specification and validation of component interaction protocols.

The management of Aussie Private Bank Limited has decided to provide services online. Due to the geographically dispersed nature of its customers, a distributed system is required. The system developers have decided to make minimal changes to the bank's existing system that was written in C. With its support for several implementation languages, CORBA is chosen as the enabling technology.

A small subset of the services in the distributed application include

- bank teller sets up customer account;

- customer makes withdrawals and deposits against his/her account.

The main type of distributed objects that provide such services to the bank customers is `Account`. The service interface for `Account` can be defined in CORBA IDL as shown in Figure 1. This interface definition presents the forms and types of some necessary operations for `Account`, but is unable to capture the precedence or order among the operation invocations. For example,

- Should `setupAccount` be always invoked before other operations on the account and not after them?

- Should the authorisation of a higher authority (i.e., another object) be required after the account is set up and before it is used?

- Should a `deposit` or `withdraw` transaction always be recorded through the relevant timestamp?

```
interface Account  {

  /* Customer information & types */
  typedef struct custInfo {
      long custID;
      string name;
      char gender;
      float bankAmt;
  };
  typedef string transID;
  typedef string timeStamp;


  /* The following operation allows
     the teller to set up an account
     for the customer. */
  void setupAccount(in custInfo cust);


  /* The following operations allow
     deposit and withdrawal of money
     by a customer. Return value is
     the ID of the transaction. */
  transID deposit(in float amount,
     out timeStamp tStamp);
  transID withdraw(in float amount,
     out timeStamp tStamp);


  /* The following operation records
     the time stamp for a particular
     transaction such as deposit or
     withdraw. */
  void recTransTimeStamp(in transID tID,
     in timeStamp tStamp);
};
```

Figure 1. Interface definition for `Account`

These are some of the questions regarding the system design that will affect the correct implementation and use of `Account`. In particular, the second question suggests that operation invocations on one object may depend on operation invocations on another object. That is, once an account is set up by a teller, the bank may require that the account be activated by the supervisor (represented by another object of the type `Supervisor`) before any transactions can be carried out on the account.

The above problems highlight the need to specify clearly the temporal relationships between operation invocations on objects. Such specification will help the clients to use the object services in an appropriate manner, as well as assisting the component developers to implement the services correctly. It can be achieved by enriching the CORBA IDL with interaction protocol specifications. A related issue is

IEEE
COMPUTER
SOCIETY

^	Upon creation of component. This refers to the event that creates the object.
an operation	Upon invocation of the operation.
an event	Upon firing of a CORBA event. This action refers to sending or receipt of a CORBA event.
$attribute$.SET($value$)	The operation that sets the $attribute$ with the $value$.
$attribute$.GET	The operation that retrieves the value of the $attribute$.

Table 1. Actions in interaction constraints

A PRECEDES B	An occurrence of B *immediately* follows an occurrence of A.
A BEFORE B	All occurrences of B (if any) follow all occurrences of A, with possible occurrences of other operations in between.
A LEADSTO B	One or more occurrences of A must lead to one or more occurrences of B eventually, with possible occurrences of other operations in between.
A PAIRWISEBEFORE B	Similar to BEFORE but the numbers of occurrences of A's and B's are taken into account while allowing the interleaving of A-B pairs.
A PAIRWISELEADSTO B	Similar to LEADSTO but the numbers of occurrences of A's and B's are taken into account while allowing the interleaving of A-B pairs.

Table 2. Temporal operators in interaction constraints

how to ensure that all the objects of a distributed system in fact interact properly at run-time. One way to address this issue is to check or validate that the actual object interactions at run-time conform to the defined interaction protocols. These two issues underlie our work in specifying and validating object interactions for CORBA systems.

3. Specifying component interactions

Realising the limitations of commercial IDLs, we have proposed a comprehensive framework of rich interface definition for software components in [13]. It incorporates

- enhanced interface signature, highlighting the need for specifying the required operations of components;

- formal specification of operation semantics;

- definition of interface configurations, reflecting the connections with neighbouring components in various use contexts;

- specification of component interaction protocols;

- characterisation of quality properties, such as those concerning the performance, security and reliability of component services.

These aspects can be used to enhance commercial IDLs to achieve better support for component based development. This paper focuses on the issues relating to component interaction protocols and their use in CORBA systems. In the remainder of this section we give an overview of our approach to interaction protocol specification. Further details can be found in [14].

In general, component interaction protocols refer to the relative sequencing of messages (order of operation invocations) going in and out of a component. In specifying the interaction protocols of a component, we have adopted a temporal logic based approach, where we use well-known temporal operators to define the temporal relationships between operation invocations. As such, the interaction protocols of a component are defined as a set of *temporal constraints*.

A temporal relationship between operation invocations is expressed in terms of the relative order between them. In general, the definition of a temporal constraint takes the following form

$$action\ tr\ action;$$

where *action* can be any of the items listed in Table 1, and *tr* is a temporal operator identifying the temporal relationship between the actions concerned. Table 2 shows a subset of temporal operators we have considered, in the form of "*A tr B*".

```
COMPONENT Account {

  setupAccount BEFORE (deposit,
     withdraw, recTransTimeStamp);

  setupAccount PRECEDES
     Supervisor.activateAccount;

  (deposit|withdraw) PAIRWISELEADSTO
     recTransTimeStamp;
};
```

Figure 2. Interaction constraints for ACCOUNT

Temporal interaction constraints can be divided into two categories: general and role-based. General constraints refer to constraints concerning only the services provided by the principal component, whereas role-based constraints also involve services supplied by one or more neighbouring components. For example, we may have the following constraints for an Account object:

- "setupAccount BEFORE deposit"
  This is a general constraint because both operations setupAccount and deposit are those of the principal component of the Account type. This constraint warrants that setupAccount is invoked before any deposit operations.

- "setupAccount PRECEDES
  Supervisor.activateAccount"
  This is a role-based constraint as its second operation, activateAccount, is that of a neighbouring component of the type Supervisor[2]. This constraint states that setupAccount must be immediately followed by activateAccount. A more detailed discussion of role-based constraints can be found in [14].

In incorporating interaction constraints into CORBA, we choose to define interaction constraints separately from existing CORBA interface definitions for simplicity. For each interface definition, we have a COMPONENT definition containing all the constraints on the corresponding type of objects. Figure 2 shows a component definition for Account, containing three constraints that answer the three motivating questions in section 2. Note that we have used some shorthand in the constraint definitions. For example, "A tr (B, C)" stands for two constraints "A tr B" and "A tr C". The first constraint states that the setupAccount operation should

---

[2]Assume Supervisor is another object type defined elsewhere with an operation named activateAccount. This operation will activate the account for deposits and withdrawals.

be invoked before any of the deposit, withdraw and recTransTimeStamp operations (and not after them). The third constraint states that each deposit or withdraw operation invocation must have a following recTransTimeStamp operation invocation.

4. Validating component interactions

Explicit specification of component interaction constraints helps the component developer and user to implement and use a component properly. Whether or not the component services are *actually* used properly at run-time is a different question. Validation or testing is often required. In this section, we introduce a framework that allows us to validate run-time interactions of a component against its defined interaction constraints.

4.1. The validation framework

Our validation framework adopts a *monitoring* approach to observe and validate a component's interactions against its interaction protocol specification. The monitoring and validation process is automated in a prototype tool RIDL-MON (RIDL Monitor), and is achieved by

- translating the constraint specifications into extended finite state machines (eFSMs) that serve as the constraints' internal representation in the monitoring tool for easy processing;

- identifying and intercepting the run-time interactions needed for validation;

- checking the intercepted run-time interactions against the constraints' internal representations, and reporting violations (if any).

Figure 3 shows the overall monitoring architecture for a distributed CORBA system. The monitored programs are instrumented with CORBA portable interceptors [25]. These interceptors are programmed to observe the incoming and outgoing operation invocations of objects, and to forward them to RIDLMON. RIDLMON translates the specified interaction constraints into eFSMs, and validates the intercepted operation invocations against the relevant eFSMs. When violation occurs, the programmer is alerted the situation and the programs are terminated. In addition, the invocation sequences of objects are captured and are available for further analysis by the programmer.

In the remainder of this section, we discuss in turn constraint representation, violation checking, and implementation of the validation tool. Further details can be found in chapters 4-6 of [19].
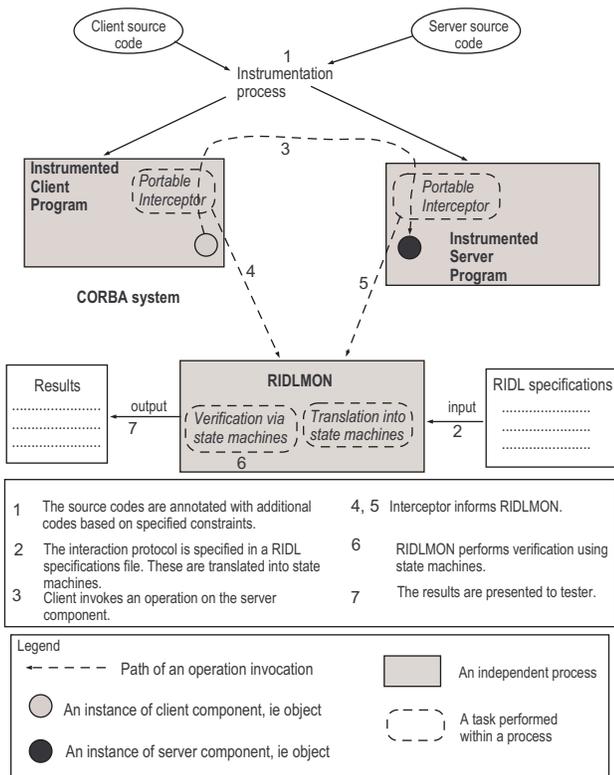
COMPUTER SOCIETY

Figure 3. The validation framework

Figure 4. eFSM representation of "`A BEFORE B`"

4.2. Constraint representation

The use of temporal constraints in specifying component interaction protocols has proved to be *intuitive* and *incremental*, and is well suited to the needs of system designers [14]. However, they are not easy to use in the run-time automatic checking of interaction constraints. As such, we choose to use an extended form of finite state machines (eFSMs) as the internal representation of temporal constraints. In general, each binary temporal relationship (operator) has a corresponding eFSM representation. In particular, PAIRWISELEADSTO and PAIRWISEBEFORE require the extended form of FSM as they need explicit assertion checking at certain state transitions (see below). Due to space limitation, we present below as examples the eFSM representations of the BEFORE and PAIRWISELEADSTO operators. The eFSM representations of other temporal operators can be found in [19].

BEFORE. The constraint "`A BEFORE B`" states that B can happen only if A has happened. Figure 4 presents the equivalent eFSM in a graphical form, where O stands for any operation other than A or B, and *E* stands for the end of interaction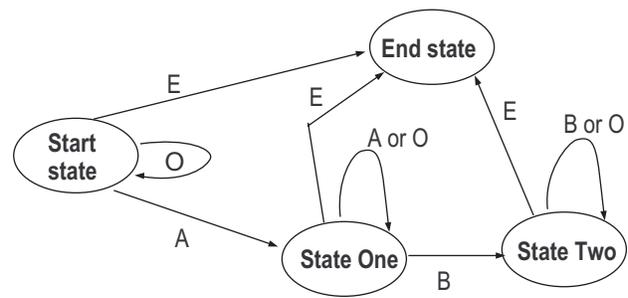s. Note that operations that are not mentioned at a given state are rejected, representing invalid operation invocations. For example, the invocation sequences AAOOAA*E* and OAOAAB*E* are valid, but OABOAO*E* is not.

PAIRWISELEADSTO. The constraint "`A PAIRWISELEADSTO B`" states that every occurrence of A must be eventually followed by a distinct occurrence of B. Figure 5 presents its eFSM representation. Note that the matching process discards the excessive B's in every maximal sub-sequence starting with A and ending with B or O. This is realised by setting the number of B's to the number of A's minus one at the reset point. Figure 6 illustrates part of the matching process for the valid invocation sequence AABBBAABBO*E*.

The number of eFSMs for a CORBA system. The validation of interaction protocols for objects in a CORBA system is based on object instances while the interaction constraints are specified for object types or interfaces. According to the above representation scheme, there will be one eFSM for each simple binary constraint of a given object instance. For example, there are 5 eFSM's for *each* object instance of Account (Figure 2): three for BEFORE, one for PRECEDES, and one for PAIRWISELEADSTO. We note that it is possible to merge the eFSMs for an object (type or interface) to reduce the number of eFSMs required. However, it may render the constraints less understandable during system testing. Consequently, we have not adopted the option of merging eFSMs in our approach.

RIDLMON is responsible for managing all the eFSMs for all object instances. In fact, the programmer can selectively decide which objects to monitor dynamically. The programmer can also switch on or off a specific constraint. As such, the programmer can incrementally test the system relative to individual objects and constraints.
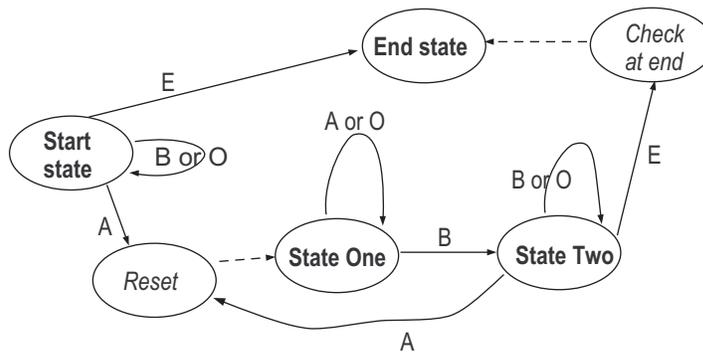
Figure 5. eFSM representation of "A PAIRWISELEADSTO B"

Valid example : AABBBAABBOE

Input	State transited to	Counters	Comment
A	Reset	countA = 1, countB = 0	In segment one
	State One	countA = 1, countB = 0	In segment one
A	State One	countA = 2, countB = 0	In segment one
B	State Two	countA = 2, countB = 1	In segment one
B	State Two	countA = 2, countB = 2	In segment one
B	State Two	countA = 2, countB = 3	In segment one
A	Reset	countA = 3, countB = 2	In segment two
	State One	countA = 3, countB = 2	In segment two

Figure 6. A matching process of PAIRWISELEADSTO

4.3. Violation checking

As discussed above, the interaction constraints are internally represented as eFSMs in RIDLMON. When an object is created, the relevant eFSMs for it are initialised. When an operation invocation is captured by the interceptors attached to the object, it is forwarded to RIDLMON to advance the eFSMs. In advancing the eFSMs, violations to the constraints can be determined. The algorithms involved are straightforward and are omitted from this paper due to space limitation. In general, a constraint violation occurs when one of the following situations is present:

- *The captured operation or event does not represent a valid transition of an eFSM.* For example, B is received at the "Start state" of the BEFORE machine (see Figure 4).

- *The captured operation or event causes a matching process to fail.* For instance, the number of A's is more than the number of B's when the PAIRWISELEADSTO machine transits to the "End state" (see Figure 5). This is applicable only to liveness constraints like PAIRWISEBEFORE and

PAIRWISELEADSTO. These liveness constraints can only be fully checked at the end of interactions.

We note that an operation invocation is usually relevant to two objects: the caller (as outgoing call) and the callee (as incoming call). Therefore, it can be captured at multiple intercepting points as actions of different objects. This is so, because our approach to monitoring is object/component specific and at a given moment the monitoring may focus on only the object at one end and not the object at the other end.

Besides normal remote operations such as withdraw and deposit of Account, Table 1 also highlights other kinds of "special" actions such as CORBA events, the get and set operations of an object's attributes. The process used to validate these actions are the same as the one used for normal operation invocations except that each action type is interpreted differently by our interceptors. We leave this issue aside because its intricacies are fairly complex to permit a satisfactory treatment in this paper. Interested readers can refer to [19].

As our approach to interaction monitoring and checking is object/component specific, we focus on only the relationships of the direct incoming and outgoing calls of an individual object. Consequently, there is not the con-
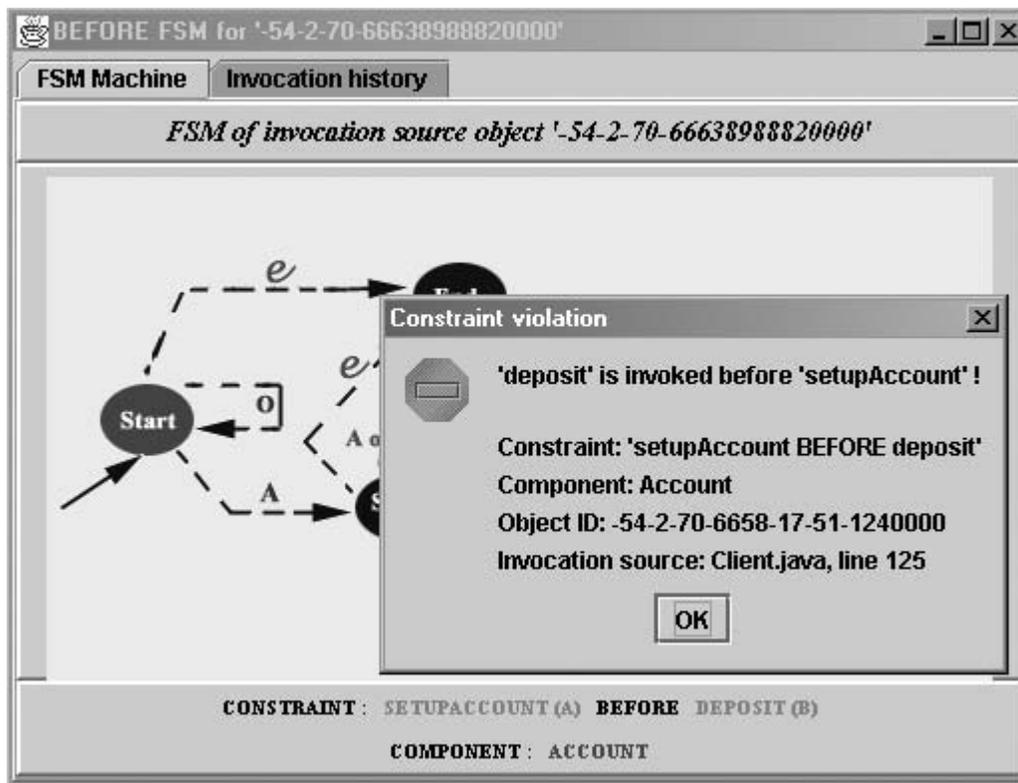
Figure 7. Constraint violation information of an application under testing

cept of nested calls and their tracing. As such, one centralised server (RIDLMON) for violation checking is sufficient and convenient to use by a programmer. Another consequence is that the checking of system-wide properties such as deadlock-free is not the concern of this research.

4.4. Implementation of the validation tool

The implementation of our validation framework involves primarily two parts: the monitor tool RIDLMON and source code instrumentation. RIDLMON implements the interceptors, the automatic translation of constraints (stored in RIDL files) into extended finite state machines (in their table form), and the automatic checking of intercepted operation invocations against these machines. RIDLMON is written entirely in Java using the Java Development Kit (JDK) for version 1.4.0 standard edition (J2SE) [32], which provides the CORBA Portable Interceptor libraries. The ORB product used throughout the project is ORBacus 4.0.5 by IONA [17].

For a given CORBA application, its programs are instrumented and compiled before being executed in the tool's monitoring environment. During program execution, the programmer can view the intercepted operation invocation sequences and the progression of the relevant state ma-

chines in graphical form. When a constraint violation is detected, the tool displays the relevant details concerning the violation. Figure 7 shows the details of a violation presented to the programmer.

In general, the implementation of RIDLMON is relatively straightforward once the internal eFSM representations for the various temporal operators are worked out. The programming of the CORBA interceptors is also easy to do with the support of the CORBA standard. We note that the programmed interceptors are generic and independent of individual CORBA applications. The main challenge in the implementation is the instrumentation of CORBA application programs, relating to object attributes and events. It involves the scanning/parsing of programs to identify the locations for instrumentation. Further details can be found in [19].

5. Related work and discussions

In our approach, the specification of interaction protocols uses a small set of temporal operators, and takes the form of constraints on the relevant interface signature elements. As such, the description of interaction protocols is very intuitive and easy to follow. In addition, it can be done in an *incremental* manner, i.e., there is no need for *up-front*

full description of a component's interaction protocols. The component's interaction constraints can be added as they are discovered. This has led our approach to be of immediate practical use, as testified by the component-based development of a telecommunications system (see below). Furthermore, the validation tool RIDLMON is fully automatic and has proved to be a very practical tool for testing run-time component interactions against specified constraints.

There have also been a number of other efforts in introducing interaction protocols into component interface descriptions. One class of approaches use finite state machines (FSMs) as the basis of protocol specification and checking. These approaches include those proposed by Yellin and Strom [36], Cho [8], and Reussner [30]. In general, these approaches base the protocol specification on a set of abstract states, and the execution of the component services is modelled as transitions between the states. The abstract states are not part of the component interface and do not have immediate meaning to the user. Therefore, this style of specification is difficult to understand from the user's perspective. For the component developers, on the other hand, it requires either the full description of the component's interaction protocols, which has proven to be impractical in real-life projects [14], or the merge of successive state machines to present a coherent picture, which makes it even more difficult to understand.

Another class of approaches base the specification of component interaction protocols on the use of process algebras like CSP and π-calculus, including those proposed by Canal et al [7], and Allen and Garlan [2]. Plasil and Visnovsky use regular expressions for specifying interaction protocols [27]. While these approaches mostly rely on the elements in the component interface for protocol specification, it is also difficult to achieve incremental specification of protocols as in the case for FSM-based approaches.

The approach proposed by Bastide et al [4] uses Petri nets to specify the behaviour of CORBA objects. The specification does not distinguish the internal semantics and the external behaviour of the component, and makes the separate description of interaction protocols and the user's understanding of the component difficult to achieve. The work by Borgida and Devanbu uses description logics to specify certain behavioural and interaction semantics of component interface [6]. It does not distinguish interaction protocols from other semantic behaviour of the component, and does not discuss how the approach can be used in practice.

Lea and Marlowe's PSL essentially uses two temporal operators (similar to our PAIRWISEBEFORE and PAIRWISELEADSTO) and some other associated constructs to specify component interaction protocols [20]. However, it does not address the issue of using these protocol constraints to validate run-time system interactions.

Closest to our work is the MOTEL effort [21]. MO-

TEL is a prototype tool for monitoring and testing communication services in CORBA applications. The communication properties of the system objects are specified using linear temporal logic. The run-time system communication events are monitored and checked against the system property specifications. We believe that from the software engineer's viewpoint, our use of temporal operators in specifying interaction protocols is much simpler and easier than the MOTEL's use of a temporal logic. In terms of monitoring, MOTEL deals with a wide range of events at the levels of objects, processes, threads and system. In contrast, the monitoring in our approach is only dependent on events at the object level as captured by the standard CORBA portable interceptors. This has dramatically simplified the design of the validation tool.

Temporal logic is also used to specify the temporal behaviour and operation semantics of CORBA objects in [18]. It is done with the aim of testing individual CORBA objects, rather than the interactions between them.

Work in the area of object-based coordination and synchronisation also concerns interactions between objects. But, it takes a prescriptive approach, i.e., enforcing interaction requirements between objects from a system design perspective, instead of checking their compatibility. Aksit et al propose to use composition filters to coordinate interactions between objects [1]. It requires the development of composition filters and abstract communication types in a procedural manner, rather than in a declarative style. Work on coordination languages by Agha and others is based on the actor model [11, 29, 24, 31, 3]. In specifying coordination constraints between objects, these languages either adopt a procedural approach and are indirect in the sense of using internal variables and calculations, or are overly simplistic in using a single real-time measure. Besides, it is unclear how practical it is to incorporate such an approach into an industrial middleware platform like CORBA.

More recently, there has been much interest in the area of Web service composition, concerning interactions of Web services. Efforts include the Web Service Conversation Language (WSCL) [16], the Web Service Choreography Interface (WSCI) [34], the Business Process Execution Language for Web Services (BPEL4WS) [33], Self-Serv [10, 5, 12], SWORD [28], Golog [22], and Mecella et al [23]. From the viewpoint of service interaction, these approaches essentially apply existing techniques to the Web service context. The techniques include procedural and logic programming, process algebra, state machines, statecharts, Petri nets and expert systems. The relevant issues of interest are very much within the scope as discussed above.

The FRIENDS project introduces a framework for integrated engineering and deployment of services [9]. It focuses on the monitoring of component interactions in CORBA systems, and allows the tester to specify entities

for monitoring and testing at run-time. Both FRIENDS and our work concern component interactions. However, the purpose of the FRIENDS framework is to provide a testing environment for CORBA applications while we focus on the validation of component interactions against predefined properties. For instance, FRIENDS captures features of primitive object interactions such as the parameter values of remote operations. But it does not consider validating interaction properties.

Finally, our work addresses component interactions at the application level, and is aimed at application developers rather than middleware designers. As such, the system level protocols implemented as part of the middleware such as CORBA services are outside the scope of this paper.

Further remarks. In developing the specification notation for interaction protocols, we emphasize its practical usability over its expressiveness, and have taken an incremental approach. We have first introduced the simple temporal operators as discussed in this paper, and then other operators such as alternation, distance and mutual exclusion. A key criteria in deciding the inclusion of a particular operator is that it should be automatically checkable and therefore can be included in an automatic verification or validation tool such as RIDLMON. To date, we have considered interaction constraints that treat operations as atomic actions. We are currently extending the notation for dealing with advanced operation sequencing issues: operation parameters and results, component concurrency, and synchronisation modes of interactions.

Our approach to interaction protocol specification has been developed, used and refined in the development of a complex telecommunications system (about 200 manyears). Compared with previous attempts at the system architecture, the adoption of component-based design with RIDL in the project has clarified the interfaces between system components, reduced the interactions between the development teams responsible for the components, and eased the system integration and testing task.

6. Conclusions and future work

The proper use of software components in a distributed system is critical to the correct functioning of the system. This is particularly so when the components are developed by third parties or different teams, and calls for richer interface description than that allowed by the IDLs of commercial middleware standards such as CORBA. In this paper, we have introduced an approach to describing and validating the interaction protocols of software components in a distributed system setting, to ascertain that the components are used as intended. The component interaction protocols are described as interaction constraints, using a small set of temporal operators. This allows intuitive and incremental description of interaction protocols, and is easy for practitioners to learn and use.

The validation tool RIDLMON monitors the run-time interactions between the components in a distributed CORBA system, and checks them against the predefined interaction constraints of the components. Any violation to the constraints represents mis-use of the relevant components, and can be identified by the validation tool. The validation tool translates the interaction constraints into extended finite state machines for easy manipulation. The component interactions at run-time are intercepted through the use of CORBA portable interceptors, and are passed on to the RIDLMON checker for validation against the extended finite state machines. In effect, RIDLMON becomes a useful tool for testing component interactions in CORBA systems, and helps to ensure the interoperability of CORBA components. We note that our approach is readily applicable to any other IDL-based object/component systems.

As mentioned earlier, we are currently investigating a number of improvements to our approach and the validation tool RIDLMON. One issue is to allow the specification of finer-grained interaction constraints involving operation parameters. Another issue is to deal explicitly with concurrency of components and synchronisation modes of component interactions.

References

[1] M. Aksit et al. Abstracting object interactions using composition filters. In *Proceedings of 1993 European Conference on Object-Oriented Programming*, pages 152–184, Kaiserslautern, Germany, July 1993.

[2] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.

[3] M. Astley and G. Agha. Customization and composition of distributed objects: Middleware abstractions for policy management. In *Proceedings of 1998 ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 1–9, Lake Buena Vista, Florida, USA, 1998.

[4] R. Bastide, O. Sy, and P. Palanque. Formal specification and prototyping of CORBA systems. In *Proceedings of the 13th European Conference on Object-Oriented Programming*, pages 474–494, Lisbon, Portugal, June 1999.

[5] B. Benatallah, M. Dumas, Q. Zheng, and A. Ngu. Declarative composition and peer-to-peer provisioning of dynamic Web services. In *Proceedings of 18th International Conference on Data Engineering (ICDE'02)*, San Jose, USA, February 2002.

[6] A. Borgida and P. Devanbu. Adding more "DL" to IDL: Towards more knowledgeable component inter-operability. In *Proceedings of the 21th International Conference on Software Engineering*, pages 378–387, Los Angeles, USA, May 1999.

[7] C. Canal, E. Pimentel, J. Troya, and A. Vallecillo. Extending CORBA interfaces with protocols. *The Computer Journal*, 44(5):448–462, October 2001.

[8] I. Cho. A framework for the specification and testing of the interoperation aspect of components. In *Object Interoperability : ECOOP'2000 Workshop on Object Interoperability*, pages 53–64, Sophia Antipolis, France, June 2000.

[9] N. Diakov, H. Batteram, H. Zandbelt, and M. van Sinderen. Monitoring of distributed component interactions. In *Proceedings of the 7th International Conference on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 227–240, Enschede, Netherlands, October 2000. Springer.

[10] M.-C. Fauvet, M. Dumas, B. Benatallah, and H.-Y. Paik. Peer-to-peer traced execution of composite services. In *Proceedings of TES 2001: LNCS-2193*, Rome, Italy, September 2001.

[11] S. Frolund and G. Agha. A language framework for multi-object coordination. In *Proceedings of 1993 European Conference on Object-Oriented Programming*, pages 346–340, Kaiserslautern, Germany, July 1993.

[12] R. Hamadi and B. Benatallah. A Petri net-based model for Web service composition. In *Proceedings of 14th Australasian Database Conference*, Adelaide, Australia, February 2003.

[13] J. Han. A comprehensive interface definition framework for software components. In *Proceedings of the 1998 Asia-Pacific Software Engineering Conference*, pages 110–117, Taipei, Taiwan, December 1998. IEEE Computer Society.

[14] J. Han. Temporal logic based specification of component interaction protocols. In *Object Interoperability : ECOOP'2000 Workshop on Object Interoperability*, pages 43–52, Sophia Antipolis, France, June 2000.

[15] J. Hernandez, A. Vallecillo, and J. Troya, editors. *Proceedings of the 2nd ECOOP Workshop on Object Interoperability (WOI'00)*, Sophia Anthipolis, France, June 2000. http://webepcc.unex.es/juan/woi00/.

[16] Hewlett-Packard. Web Service Conversation Language (WSCL) 1.0. Technical report, W3C, March 2002.

[17] IONA Technologies. Iona website. http://www.iona.com, accessed on August 3, 2001.

[18] R. Jagannathan and P. Silvilotti. Increasing client-side confidence in remote component implementations. In *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 52–61, Vienna, Austria, September 2001. ACM Press.

[19] K. Ker. Monitoring Component Interoperation in Middleware Systems. BNetComp (Honours) thesis, Monash University, Australia, October 2001.

[20] D. Lea and J. Marlowe. Interface-based protocol specification of open systems using PSL. In *Proceedings of 1995 European Conference on Object-Oriented Programming*, pages 374–398, Aarhus, Denmark, August 1995.

[21] X. Logean. *Run-time Monitoring and On-line Testing of Middleware Based Communication Services*. PhD thesis, Ecole Polytechnique Federale De Lausanne (Lausanne EPFL), 2000.

[22] S. McIlraith and T. Son. Adapting Golog for composition of semantic Web services. In *Proceedings of 8th International Conference on Knowledge Representation and Reasoning*, Toulouse, France, April 2002.

[23] M. Mecella, B. Pernici, and P. Craca. Compatibility of e-services in a cooperative multi-platform environment. In *Proceedings of TES 2001: LNCS-2193*, Rome, Italy, September 2001.

[24] B. Nielsen, S. Ren, and G. Agha. Specification of real-time interaction constraints. In *Proceedings of the First International Symposium on Real-Time Computing*, pages 206–214, 1998.

[25] OMG. Portable interceptors. ftp://ftp.omg.org/pub/docs/orbos/99-12-02.pdf, December 1999.

[26] OMG. CORBA success stories. http://www.corba.org/success.htm, last modified May 30, 2001, accessed July 16, 2001.

[27] F. Plasil and W. Visnovsky. Behaviour protocols for software components. *IEEE Transactions on Software Engineering*, 28(11):1056–1076, November 2002.

[28] S. Ponnekanti and A. Fox. SWORD: A developer toolkit for Web service composition. In *Proceedings of 2002 World Wide Web Conference*, Honolulu, USA, May 2002.

[29] S. Ren and G. Agha. RTsynchronizer: Language support for real-time specifications in distributed systems. In *Proceedings of ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems*, pages 50–59, La Jolla, CA, USA, June 1995.

[30] R. Reussner. An enhanced model for component interfaces to support automatic and dynamic adaption. In *Object Interoperability : ECOOP'2000 Workshop on Object Interoperability*, pages 33–42, Sophia Antipolis, France, June 2000.

[31] D. Sturman and G. Agha. A protocol description language for customizing failure semantics. In *Proceedings of the 13th Symposium on Reliable Distributed Systems*, pages 148–157, October 1994.

[32] Sun Microsystems, Inc. Java 2 platform, standard edition v1.4.0 (J2SE). http://java.sun.com/j2se/1.4/, September 2001. Last accessed October 5, 2001.

[33] The BPEL4WS Team. Business Process Execution Language for Web Service (BPEL4WS). Project report, www-106.ibm.com/developerworks/library/ws-bpel/, May 2003.

[34] The WSCI Team. Web Service Choreography Interface (WSCI) 1.0. Technical report, W3C, August 2002.

[35] A. Vallecillo, J. Hernandez, and J. Troya, editors. *Proceedings of the 1st ECOOP Workshop on Object Interoperability (WOI'99)*, Lisbon, Portugal, June 1999. http://polaris.lcc.uma.es/ av/ecoop99/ws/.

[36] D. Yellin and R. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, March 1997.