

Available from: http://dx.doi.org/10.1007/978-3-540-24679-4_17

Copyright © Springer-Verlag Berlin Heidelberg 2004.
This is the author’s version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library. The definitive version is available at http://www.springerlink.com.
Interaction Compatibility: An Essential Ingredient for Service Composition

Jun Han

School of Information Technology
Swinburne University of Technology
John Street, Hawthorn, Vic. 3122, Australia
email: jhan@it.swin.edu.au

Abstract. Common to Grid services, Web Services, software agents and software components is that they are independently built and provide services aimed for composition. A key issue is whether or not the services in a composite system can interact with each other “sensibly” and as orchestrated by the enclosing composition. In this paper, we introduce an approach where we can specify individual services’ interaction intentions, and check their compatibility in a composite system. We discuss the use of the approach, the specification language, and the compatibility checking tool in the context of software components. They are equally applicable to other service frameworks mentioned above.

1 Introduction

In recent years there has been great interest in service-oriented systems. These include systems built from Grid services, Web services, software agents or software components. The key driving force behind all these efforts is the potential to deliver systems with resource sharing/reuse, higher quality, lower cost and shorter time-to-market. This potential rests on the ability of assembling composite systems from independently developed services.

While having their own specific issues to address, the above mentioned service frameworks share many common features and challenges. One of the key challenges is how to ensure that the services used in a system can work together effectively to achieve the system goals. In particular, there are two specific issues of concern:

1. Are services compatible with each other when they interact in the composite system?
2. How to ensure that the services work collaboratively towards the overall system goals.

The first issue is about whether or not the interacting services inherently satisfy (i.e., are compatible with) each other in terms of functionality, interaction and quality of service. The second issue is about how to coordinate the various services to meet the system requirements. The two issues are inter-related. This paper concerns the first issue, service compatibility.
Without compatibility, the services involved in a composite system can not work together properly (if at all), not to mention achieving system goals. In general, services play provider and consumer roles relative to each other in the composite system. Compatibility exists at different levels. First, the service providers should have the functionality that the service consumers require. This requires that the corresponding required and provided services “match” syntactically and semantically. Second, a service provider usually requires and provides a number of operations and has a particular intention implemented in regards to how the operations interleave to effect the services it provides. We refer to the intended (partial) order of operation invocations (or events) as the service’s interaction protocols. The interaction protocols of related service providers and consumers should also match. Third, a service consumer may demand a service provider to guarantee certain qualities in providing a service while the provider requires the consumer to satisfy certain obligations. For example, a tax lodgement service (as a service consumer) may require that the tax calculation service (as a service provider) to encrypt the tax calculation results while the calculation service requires the lodgement service to encrypt the tax return data in the first place. The quality of service may concern security, reliability, performance, etc. In general, service compatibility at all levels are essential. This paper is particularly concerned with compatibility in service interaction protocols.

To facilitate interaction compatibility, we need to be able (1) to specify clearly the interaction intentions of individual services – specification, (2) to check statically at assembly/composition time the compatibility of the related services’ interaction protocols – verification, and (3) to check dynamically at run-time that the actual service interactions conform to the interaction protocols – validation. In this paper, we introduce a notation for specifying service interaction protocols and a validation tool for checking run-time conformance of service interactions against predefined protocols, using the CORBA object platform as an example service framework. They can be equally applied to other service frameworks. 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 examines the requirements for interaction compatibility in service composition. Section 3 introduces our approach to interaction compatibility, including the specification and run-time validation of service interactions. The service interaction protocols are specified as temporal constraints. The run-time validation involves the interception of service interactions, the FSM-based internal representation of temporal constraints, and the conformance checking of service interactions against the temporal constraints. Section 4 discusses related work before section 5 summarises the key contributions and future directions.

2 Requirements for Interaction Compatibility

Service-oriented computing involves (1) independently developed services and (2) compositions of these services to form systems of specific purposes. The
services used in a given composite system interact with each other to achieve system goals. It is important that the services used in the system are compatible with each other. To gain proper understanding of the services and facilitate system composition, the commonly adopted approach is to publish or specify the externally observable properties of the services in their interfaces and check their compatibility in the context of a given composite system. In general, the interface properties of a service should include those about its functionality, interaction and quality. Only through the publication, understanding and checking of these properties can a service be possibly used properly, especially in the context of dynamic composition.

The existing models for service interface specification, such as those of CORBA, EJB, COM/.NET and WSDL, have mainly focused on specifying the syntax and types of service operations. More recently, we have seen increased interest in interface specification and checking for interaction and quality properties of services.

In the context of a composite system, the interaction compatibility between the services involved is critical to the proper functioning of the system. Even if the services provide each other with expected functionality and quality of service, the system may still not function as required. For example, an on-line auction system may involve three types of services: the auctioneer service, the bidder service, and the banking service. A bidder interacts with the auctioneer to bid for items on sale, and both the auctioneer and the bidders interact with the banking service to open accounts and settle financial transactions. The banking service naturally requires that an account be set up before any operations on the account. The auctioneer service requires that a bidder first register with the auctioneer, including the account details as opened with the banking service, before accepting any of its bids. Furthermore, the auctioneer checks with the banking service that a bidder has sufficient fund in its nominated account before acknowledging a bid. While having the functionality of lodging bids, the bidder service may or may not be able to conform the auctioneer’s requirements of registration and sufficient fund. If not, the auctioneer and the bidder services are not compatible, which will result in system error or even system failure.

The above example has clearly shown that
1. the interaction protocols or the partial ordering of operations that a service assumes and implements need to be clearly specified for understanding and use at the time of service composition — specification;
2. at the time of composition (static or dynamic), the compatibility between the interaction protocols of the related services needs to be checked to avoid system error and failure — verification;
3. after service composition, tests can be further conducted to validate the actual service interactions against the relevant protocol specifications — validation.

To provide practical support for service composition, a simple specification notation and automatic verification and validation tools are required for specifying and checking service interaction protocols.
3 An Approach to Interaction Compatibility

In this section, we introduce an approach to the specification and compatibility checking of service interaction protocols. We illustrate its use regarding interaction specification and validation in the context of software components with CORBA as the service platform (see [8] and [9] for further details). The approach is equally applicable to other service frameworks like Web services and Grid services.

3.1 Specifying Service Interactions

Realising the limitations of current commercial approaches to service interface definition, we have proposed a comprehensive framework for rich interface definition in the context of software components in [7]. It deals with service functionality, interaction and quality. In this paper, we focus on the issue of component interaction protocols and their compatibility. 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

\[
\text{action} \; tr \; \text{action};
\]

where \textit{action} can be an operation, an event, a get or set operation on an attribute, or the creation of a component; \textit{tr} is a temporal operator identifying the temporal relationship between the actions concerned. Some of the temporal operators are \texttt{PRECEDES}, \texttt{BEFORE}, \texttt{LEADSTO}, \texttt{PAIRWISEBEFORE} and \texttt{PAIRWISELEADSTO}.

In incorporating interaction constraints into CORBA, we have chosen to define interaction constraints separately from existing CORBA interface definitions for simplicity. For each CORBA interface definition, we have a \texttt{COMPONENT} definition containing all the constraints on the corresponding type of objects. Figure 1 shows a component definition for \texttt{Auctioneer}. The \texttt{Auctioneer} provides the \texttt{register} and \texttt{bid} operations, and requires an \texttt{acknowledge} operation from the \texttt{Bidder} components and an \texttt{enoughFund} operation from the \texttt{Bank} component. The \texttt{Auctioneer} interface/component has three interaction constraints. The first constraint states that for a given bidder, the \texttt{register} operation should be invoked before any of the other operations (and not after them). The second constraint states that a \texttt{bid} must be immediately followed by a call to the \texttt{bank}'s \texttt{enoughFund} operation to check if there is sufficient fund. The third constraint states that each \texttt{bid} invocation must lead to an \texttt{acknowledge} operation on the bidder.
COMPONENT Auctioneer {
    register (bidder) BEFORE (bid(bidder, ...),
        bidder.acknowledge (...),
        bank.enoughFund(bidder, ...));
    bid(bidder, ...) PRECEDES bank.enoughFund(bidder, ...);
    bid(bidder, ...) PAIRWISELEADSTO bidder.acknowledge(...);
};

Fig. 1. Interaction constraints for Auctioneer

3.2 Validating Service 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 regard, we have implemented a validation tool for checking the run-time interactions of a component against its defined interaction constraints. The tool RIDLMON (Rich IDL Monitor) adopts a monitoring approach, and achieves protocol validation 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 interactions against the constraints’ internal representation, and reporting violations (if any).

RIDLMON is written entirely in Java using JDK 1.4 for the CORBA product ORBacus 4.0.5. For the purpose of intercepting run-time interactions between components, we have used CORBA portable interceptors [11].

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 [8]. 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, PAIRWISEBEFORE and PAIRWISELEADSTO require the extended form of FSM as they need explicit assertion checking at certain state transitions. For example, Figure 2 presents the eFSM representation of “A PAIRWISELEADSTO B”, where 0 stands for any operation other than A or B. Note that the matching process discards the excessive B’s in every maximal sub-sequence starting with A and ending with B or 0, by setting the number of B’s to the number of A’s minus one at the reset point. At the end, the number of A’s must be less than or equal to the number of B’s.
Fig. 2. eFSM representation of “A PAIRWISELEADSTO B”

Violation Checking. As discussed above, the interaction constraints are internally represented as eFSMs in RIDLMON. When a component is created, the relevant eFSMs for it are initialised. When an operation invocation is captured by the interceptors attached to the component, it is forwarded to RIDLMON to advance the eFSMs. In advancing the eFSMs, violations to the constraints can be determined. 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 of the BEFORE machine.
- 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 2). 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 components: the caller (as outgoing call) and the callee (as incoming call). Therefore, it can be captured at multiple intercepting points as actions of different components. This is so, because our approach to monitoring is component specific and at a given moment the monitoring may focus on only the component at one end and not the component at the other end.

4 Related Work

There have been a number of other efforts in introducing interaction protocols into component interface descriptions. These include the use of finite state machines [16], process algebras [5, 2], regular expressions [12], Petri nets [3],
description logics [4], and temporal descriptions [10]. These approaches all have their limitations (see [9] for a detailed discussion).

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. Typical efforts include composition filters [1], and coordination languages [6, 13].

More recently, there has been much interest in the area of Web service composition, concerning interactions of Web services. Example efforts include the Web Service Choreography Interface (WSCI) [15], and the Business Process Execution Language for Web Services (BPEL4WS) [14]. From the viewpoint of service interaction, these approaches essentially apply existing techniques to the Web service context.

5 Conclusions

The proper use of services in a composite service-based system is critical to the correct functioning of the system. This is particularly so when the services are developed by third parties or different teams, and calls for richer interface description than that allowed by the interface definitions in the current standards such as CORBA IDL and WSDL. In this paper, we have analysed the requirements for interaction compatibility, and introduced an approach to describing and validating the interaction protocols of individual services. The service 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 services in a composite system, and checks them against the predefined interaction constraints of the services. Any violation to the constraints represents mis-use of the relevant services, and can be identified by the validation tool. We note that while our approach has been implemented using the CORBA object platform, it is equally applicable to any other IDL-based service frameworks.

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. Furthermore, we are also investigating static compatibility checking for service interaction protocols at composition-time.

Acknowledgment. I would like to thank Ksze Kae Ker for his effort in implementing the validation tool RIDLMON.

References

152–184, Kaiserslautern, Germany, July 1993.

This article was processed using the L\LaTeX\ macro package with LLNCS style