Science of Computer Programming 77 (2012) 870–886

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A generic framework for *n*-protocol compatibility checking

Francisco Durán [a], Meriem Ouederni [a,*], Gwen Salaün [b]

[a] *Department of Computer Science, University of Málaga, Spain*

[b] *Grenoble INP–INRIA–LIG, France*

ARTICLE INFO

Article history:
Received 2 December 2009
Received in revised form 21 March 2011
Accepted 23 March 2011
Available online 9 April 2011

Keywords:
SOC
Transition systems
Non-observable actions
Compatibility
Maude

ABSTRACT

Service-Oriented Computing promotes the development of new systems from existing services which are usually accessed through their public interfaces. In this context, interfaces must be *compatible* in order to avoid interoperability issues. In this article, we propose a new framework for checking the compatibility of *n* service interfaces. Our framework is *generic*, in the sense that it implements several compatibility notions useful for different application areas, and *extensible* since new further notions can easily be incorporated. We consider a service interface model which takes behavioural descriptions with value-passing and non-observable actions into account. Our compatibility checking framework has been fully implemented into a prototype tool which relies on the rewriting logic-based system Maude.

1. Introduction

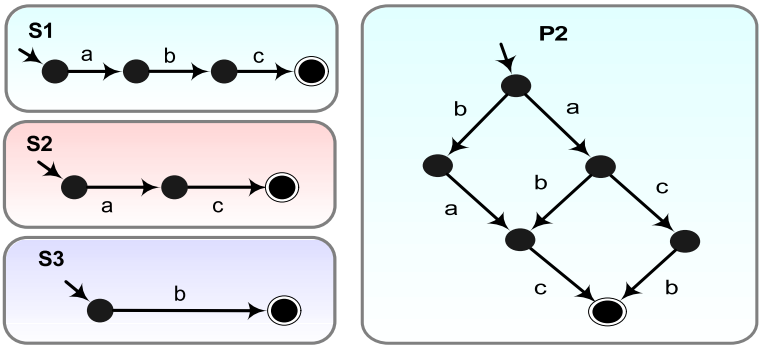
Service-Oriented Computing (SOC) aims at developing new software systems by reusing existing services. Services are software applications which are developed independently, loosely coupled, and accessed through their public interfaces. Thus, it is essential to ensure that service interfaces *fit* with each other in the system being developed. To this goal, compatibility checking is of utmost importance to guarantee that interfaces are safely reused, so that they can successfully interoperate. However, verifying compatibility is difficult, especially when the model of service interfaces takes interaction protocols (messages and their application order) into account, which allows one to avoid erroneous behaviours or deadlock situations when executing a set of services together [37].

There exists much work on the compatibility of services described using different interface models (see, for instance, [42,5,6,14] for automata, [8,15] for π-calculus, [23,28] for Petri nets, and [2,9] for state machines). Nevertheless, these proposals usually do not consider value-passing, *i.e.*, parameters coming with the messages exchanged between services, and non-observable or internal (τ) actions. Furthermore, most of the existing approaches on service compatibility rely on one particular compatibility notion defined with respect to a specific application domain (see, for instance, [28] for service composition [2,9,23], for service substitution, or [15] for service choreography), and the compatibility is often checked for two services (very few contributions on *n*-services compatibility exist; see Section 5). Finally, only a few existing proposals are equipped with appropriate tools for automating the service compatibility check.

To fill these gaps, we consider a generic approach for checking the compatibility of n ($n \geq 2$) service interfaces (signature and interaction protocols). The interaction protocols are described by means of *Symbolic Transition Systems* (STSs), and take value-passing as well as internal behaviours into account. We formalise several compatibility notions for n services, considering different strategies for handling value-passing. We propose a framework where these notions of compatibility can be automatically checked in a unified way. It is also possible to return a counterexample in order to detect

* Corresponding author. Tel.: +34 693364602; fax: +34 952131397.
E-mail addresses: duran@lcc.uma.es (F. Durán), meriem@lcc.uma.es (M. Ouederni), Gwen.Salaun@inria.fr (G. Salaün).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.03.009



(a) Service behaviours. (b) Composition of S2 and S3.

Fig. 1. *N*-service compatibility.

the incompatibility source. This framework is supported by a Maude-based implementation and can be extended with new compatibility notions and new strategies to handle value-passing.

Although studying the compatibility of n interacting protocols is much harder when $n > 2$, it is easy to find situations in which checking the compatibility of n protocols is required, *e.g.,* in the composition of several services. In what follows, we introduce an example to illustrate that the compatibility of n services cannot always be computed using techniques existing for two protocols [16,5]. Let us consider a system with three services, S1, S2 and S3, which interoperate through the exchange of messages a, b and c. Fig. 1(a) shows the service behaviours described using transition systems (presented in Section 2). We assume a synchronous binary communication model and a compatibility notion which requires that two services are compatible if they can always synchronise on each observable action (we refer to this notion as *bidirectional complementarity* or *BC* for short). A naive solution to check the compatibility of S1, S2 and S3 using two service-based techniques is as follows. We could first compose $n - 1$ services (here, $n = 3$) and then check the compatibility of the composite service with the remaining one. For instance, the composition of S2 and S3 returns the transition system P2 which consists of interleaved traces shown in Fig. 1(b).[1] Here, P2 can start by performing either a or b, whereas S1 always performs a and then b. As consequence, the protocols P2 and S1 are not *BC* compatible because they cannot synchronise on b at their initial states, *i.e.*, the message b does not have any match. On the other hand, if we consider the three services all at once, their compatibility can be checked as follows: initially, S1 and S2 can synchronise on message a, S3 waits until it can perform b with S1, and finally, the system terminates successfully once S1 and S2 synchronise on message c. Thus, the three services are compatible. This simple example illustrates the need to extend the existing compatibility notions and their checking from two to $n \geq 2$ services.

In this article, we illustrate our approach for checking n-service compatibility with some well-known compatibility notions, namely *bidirectional complementarity* [5] (*BC*), *unspecified receptions* [6,42] (*UR*), *deadlock freeness* [5] (*DF*), *one path* [15] (*OP*), and *unidirectional complementarity* (*UC*). Considering different compatibility notions makes our framework useful in different contexts, *e.g.*, in the case of the client/server model one would prefer the *UC* notion (further arguments are given throughout Section 3.3). Our proposal can also be used for several service issues, such as composition, adaptation, substitution, or discovery. For instance, in the context of service adaptation [30], a counterexample might be helpful to automate the specification of an intermediate service, *i.e.*, adaptor, which enables to continue the communication in spite of existing incompatibilities.

A prototype tool implementing our framework has been developed in the rewriting logic system Maude [10,11]. Other alternatives were studied, such as using process algebraic tools (*e.g.*, CADP [21] or MCRL2 [22]), but we found the Maude system more convenient than these tools for the development of our generic framework encoding the different compatibility definitions. We present in Section 6 a short comparison between our approach and these alternative solutions.

This article advances our previous work presented in [16] as follows:

1. We extend our approach for checking protocol compatibility from two to n services.
2. In [16], we studied the *DF*, *UR*, and *BC* compatibility notions for two services. Here, we extend these notions for n services, and also consider the *OP* and *UC* compatibility notions.
3. Here, we consider one single strategy for handling internal behaviours. We require that each execution of a τ action must lead the respective protocol into a state where the whole system remains compatible.
4. In this article, exchanged parameters can be compared with respect to different parameter strategies.
5. The prototype implementation supports these extensions, and some experimental results are discussed in Section 4. Moreover, we present how to extend our framework with new compatibility notions and strategies for dealing with value-passing.
6. We update the comparison of our proposal with related approaches.

[1] In this example, services are composed using a CCS-like semantics [33].