

Towards Dynamic Matching of Business-Level Protocols in Adaptive Service Compositions

Alan Colman, Linh Duy Pham, Jun Han, and Jean-Guy Schneider

Faculty of ICT, Swinburne University of Technology
PO Box 218, Hawthorn, 3122, Australia
{acolman, lpham, jhan, jschneider}@ict.swin.edu.au

Abstract. In a service composition, it is necessary to ensure that the behaviour of a constituent service is consistent with the requirements of the composition. In an *adaptive* service composition those behavioural requirements may be continually changing. This paper shows how the behavioural requirements in abstract service definitions (roles) can be dynamically and incrementally defined using constraints. These constraints are then used to generate finite state automata, which are used to check the compatibility of candidate services that have their behaviour expressed in static interface descriptions such as OWL-S.

1 Introduction

In addition to the development of services, Service-Oriented Computing (SOC) application development is a process consisting of service discovery, evaluation and composition. To support the composition of services, a variety of standards such as WSDL and SOAP have helped resolve the heterogeneity in implementation platforms. However, standardisations in the syntactic interface description of services (e.g. WSDL) alone are not sufficient to ensure the correct interoperation of services.

Previous work in component-based software engineering suggests that there are four levels of component interface specification: *syntactic*, *behavioural*, *synchronisation* and *QoS (Quality of Service)* [2]. In Web services, the need for semantically rich descriptions of services has resulted in a number of initiatives such as OWL-S and WSMO. In this paper we focus on *behavioural* interoperability, in particular the sequence of exchanged messages (protocols) between services. Of the above initiatives OWL-S provides explicit semantics for specifying the behaviour of services in its process model. OWL-S specifies its interaction in terms of definitions for *atomic*, *simple* and *composite processes*. WSMO, on the other hand, specifies the behavioural protocol of a service by using Abstract State Machines as the underlying formalism to represent the service's orchestration and choreography. WSMO also provides the specification of mediators to solve the mismatches at the data, communication protocol, and process levels. In contrast, rather than stipulating the existence of a new type of component in the Web services infrastructure, OWL-S provides to Web services and their clients the information that is needed to find existing mediators that can reconcile their mismatches [1].

The limitation of behavioural descriptions in OWL-S and WSMO is that they are *static* descriptions defined on the interface of a service. This is fine for individual service instances if all we are attempting to do is to match two services. To achieve interoperation of two static services at a behavioural level, the services need to share a behavioural ontology, and some mechanism needs to be provided to check the compatibility of the descriptions so that the service can interoperate. For example, recent work [4, 7, 8] addresses the problem of interoperation and matching between services that provide static behavioural descriptions.

As well as describing behaviour of the interface of a service, other approaches describe the behaviour of service *compositions*, either as orchestrations (BPEL) or choreographies (WS-CDL). However, like the above rich interface descriptions, these compositional descriptions are not designed to be dynamically generated, or incrementally altered, at runtime. The Role-Oriented Adaptive Design (ROAD) framework [3], on the other hand, does support the dynamic composition of services. In this paper, we show how the dynamic behavioural descriptions that are contained within a ROAD composite can be matched to the static behavioural descriptions presented by service interfaces in the form of OWL-S.

If we are to create dynamic compositions of services, the composition itself may have changing behavioural requirements. Consider a Library that buys books from a range of vendors. To do this the Library uses a broking service composite (Book Broker composition) that has relationships with a range of vendors. These vendors provide Web services that have various business-level rules for quoting, ordering and payment. Clearly, there are many business level protocols that govern the interactions between the buyer and the vendors. Some of these involve the defining acceptable sequences of interactions or constraints on ordering of interactions; for example, terms of payment (payment before delivery, or delivery before payment), conditions of order cancellation, non-delivery of goods, etc. As these services have been developed by different organisations, there may be mismatches between the constraints on the sequence of interactions between services (i.e. “*protocol mismatch*”).

For example, one book vendor may require payment before it delivers an order of books, while another vendor may be prepared to deliver books on receipt of an order and expect payment on invoice. The Book Broker composition has to mediate a range of interaction requirements from both the buyers and the vendors that use its service. As the Book Broker service cannot know in advance all the possible protocols that potential vendors and buyers might present, it therefore needs to be able to adapt or dynamically generate the interfaces it presents in order to interact with a range of vendors and buyer services whose interaction requirements have not been foreseen during the design phase. The protocol descriptions in these dynamically generated interfaces must be consistent with the protocols of the constituent services that are being dynamically added to the composition.

2 Dynamic Protocol Specification and Aggregation in ROAD

Our approach to creating adaptive service compositions is to use the ROAD (Role-Oriented Adaptive Design) framework [3]. In ROAD, service compositions are role-based interaction structures, where *roles are dynamic abstract service definitions* that

(among other things) define the expected behaviour of a service that is bound to that role. Services interact with each other *via* their roles. To show how protocols are represented in ROAD, we will illustrate a service composite consisting of two roles: Buyer, and Vendor. As shown in Fig. 1, different services can play a role at different times, i.e. Amazon and Barnes & Noble services are candidate players of the Vendor role.

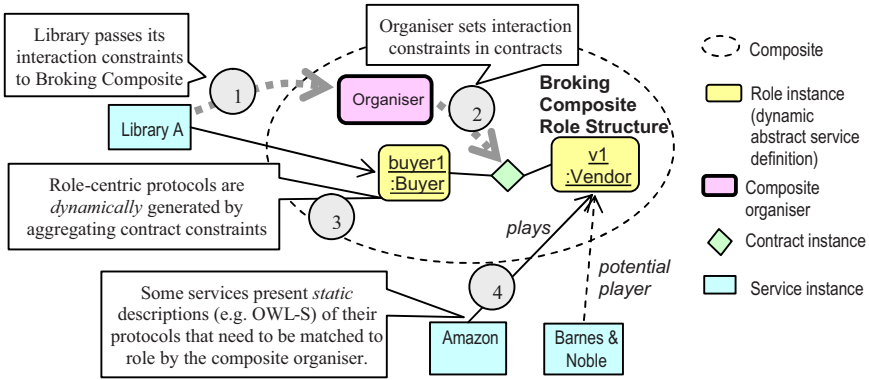


Fig. 1. Protocols in a ROAD service composite

A ROAD *contract* is a rich connector between two roles. More than just a binding, it stores the interaction constraints and provides a mechanism to intercept the messages exchanged between two roles during run-time in order to verify the interaction. The monitoring ability of ROAD contracts is discussed elsewhere [9]. Of particular relevance to the discussion in this paper, is the ability of ROAD contracts to define *protocol clauses* that describe permissible sequences of transactions that can occur between roles.

An *organiser* provides an overall management over roles and contracts within its composition. The organiser adapts the composite by creating (and destroying) roles, and creating (and revoking) contracts between these roles and the binding between roles and services. The organiser also provides a management interface that allows the non-functional requirements (i.e. behavioural or QoS requirements) to be set. For example, in Fig. 1 (step 1) the library passes its interaction constraints to the composite organiser. In order to specify the protocols, ROAD uses a *temporal constraint* language Interaction Rule Specification (IRS) [5]. IRS is used to dynamically specify the temporal constraints of the protocols between two roles. These constraints are stored in the contract between those roles. Listing 1 illustrates a (somewhat simplified) constraint specification for a Buyer-Vendor contract protocol, and shows how constraints can be incrementally modified at run-time by removing unwanted constraints and adding new constraints; unlike other approaches such as BPEL where the entire composition script has to be reworked. Our approach also provides an automatic consistency checking to ensure no violation has occurred during the changes. The reader is referred to [9] where we describe how IRS constraints can be incrementally added or deleted from ROAD contracts.

```

contract protocol BuyerVendor{
  Vendor.order precedes Buyer.orderConfirmation globally;
  Buyer.orderConfirmation precedes Buyer.receiveDelivery globally;
  // Buyer.receiveDelivery leads to Broker.receivePayment globally; // deleted
  Broker.receivePayment precedes Buyer.receivePaymentAck globally;
  Buyer.receivePaymentAck precedes Buyer.receiveDelivery globally; // newly added constraint
}

```

Listing 1. Modified temporal constraints in Buyer-Broker contract in IRS notation

In the context of maintaining valid protocol descriptions in its composite, the organiser is responsible for writing protocol constraints into the contracts it creates between the roles (Step 2). These constraints are converted into finite state automata (FSA) within the contract so that interaction can be checked at runtime, as discussed in [5, 9]. (For multiple contracts in a composite, the organiser also maintains a model of any dependency constraints between these contract protocols.) Where a role is bound to *more* than one contract (not shown in Fig 1.), these FSAs are aggregated into a single protocol description for the role (a ‘role-centric protocol’ as in step 3).

Now that we have a dynamically constructed a behavioural specification in a role’s abstract service description, the problem remains how to match concrete services with this specification (step 4 in Fig. 1). For the purposes of this discussion, we will assume that candidate services provide a description of their behaviour in OWL-S; however our general approach is not limited to any particular behavioural description. In order to match the service with its behavioural requirements as defined in the role, we need a common formalism that enables reasoning about temporal constraints. We use FSAs for this purpose because, as described in the next section, FSAs allow us to identify different types of mismatch. To convert the claimed behaviour of a service as expressed in its OWL-S description (or other similar descriptions) to an FSA, we utilise the Mindswap API [6] to parse the OWL-S files. Each atomic process is converted to a simple FSA. Following a method described by others in [7], these elementary atomic process FSAs are then composed, taking account of the control constructs of which they are a part (e.g. sequence, choice, repeat-while, repeat-until, split, or split-join). We now have FSA representations of the protocols of both the role and any candidate services that can be compared for matching.

3 Towards Resolving Protocol Mismatches

As described above, the behavioural protocols of the role (i.e. the role-centric protocol) and the service’s OWL-S files are translated into FSAs. In order to check for the compatibility of a service against a role, we extend an approach described in [5, 10], whereby the service provider can be said to *satisfy* the behavioural requirements of a role when the intersection of the role’s FSA and the complement of the service provider’s FSA is empty. While the above approach can find an *exact* match between the behavioural requirements, as defined in a role’s abstract service description and a concrete service’s OWL-S description, in many cases it may be

difficult to find a service that matches the composition's requirements exactly¹. We therefore extend this approach by matching FSAs based on the following categories:

- **fully compatible:** For every path that leads to final states in the role's FSA, there is an identical path in the service's FSA. This is not symmetrical in the sense that the service's FSA might also support other paths that do not exist in the role's FSA. However from the role's perspective, these extra paths are of no interest.
- **exactly compatible:** This is the special case of the fully compatible case where every path that leads to final states in the role's and service's FSAs are identical.
- **partially compatible:** The role's FSA has some paths that lead to final states which are identical to those in the service's FSA. Not all the paths that lead to final states in the role's FSA are supported by the service's FSA.
- **incompatible:** The role's FSA and the service's FSA do not have common paths that lead to final states.

The above categories are illustrated in Fig. 2.

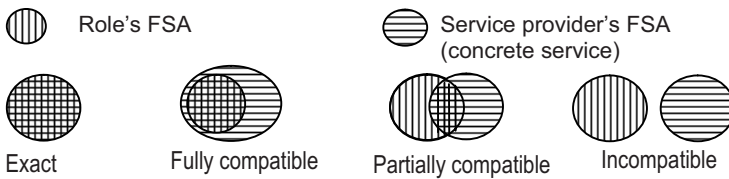


Fig. 2. Categories of protocol matching between services

To carry out the matching process, we make use of an FSA intersection algorithm. Firstly, the intersection of the role FSA and the service's FSA is found. In the case that $FSA_intersect$ does not have any path that leads to final states (i.e. the $FSA_intersect$ is *empty*), it corresponds to the 'Incompatible' case and we do not proceed any further. If the $FSA_intersect$ has some paths that lead to final states (i.e. the $FSA_intersect$ is *not empty*), these paths are evaluated to see if they are exactly the same as all the paths in role's FSA. The evaluation is done by computing the intersection of the role's FSA with the complement of $FSA_intersect$. If the resultant FSA is *empty*, it is concluded that the service FSA matches the role's requirement ('Exact' or 'Fully compatible' case). If the resultant FSA is *not empty*, it corresponds to the 'Partially compatible' case.

In the case of *full* or *exact* compatibility, the service can fulfil the role's behavioural protocol. The service will then be bound to the role as its player. In the case of 'Incompatible', the service cannot fulfil any of the role's interaction, so this service is of no interest to the composition.

¹ There exists other work [4, 8] that address the matching and ranking the compatibility at the structural level, in this paper we are concerned only at the compatibility categorisation at the behavioural level.

In the case of *partial* compatibility, the service can partially satisfy (*satisfice*) the role's behavioural protocol, i.e. there is at least one possible conversation that can take place among the role and the service. If the composition is to use this service, it needs to have an adaptation mechanism to ensure that all the conversations between the role and the service are supported by the service, i.e. only the conversations that follow the common paths are allowed. For the remaining interaction paths required by the role, the composite organiser will search for other services that can satisfy those paths. This becomes the problem of functional service composition rather than just protocol matching which we plan to address in future work.

In conclusion, it is necessary in a service composition to ensure that the behaviour of a constituent service is consistent with the requirements of the composition. In *adaptive* compositions, those behavioural requirements may be continually changing. In the context of the ROAD framework, this paper shows how the behavioural requirements in abstract service definitions (roles) can be dynamically and incrementally defined using IRS constraints. These constraints are then used to generate FSAs (finite state automata). These FSAs are then used to automatically check the compatibility of candidate services that have their behaviour expressed in static interface descriptions such as OWL-S. Further work needs to be done to address the problem of mismatches between composite behavioural requirements and the actual behaviour of services.

References

1. Ankolekar, A., Martin, D., McGuinness, D., McIlraith, S., Paolucci, M., Parsia, B.: OWL-S' relationship to selected other technologies. [Online]: <http://www.w3.org/Submission/2004/SUBM-OWL-S-related-20041122/>
2. Beugnard, A., et al.: Making components contract aware. *IEEE Computer* 32, 38–45 (1999)
3. Colman, A., Han, J.: Using role-based coordination to achieve software adaptability. *Science of Computer Programming* 64, 223–245 (2007)
4. Jaeger, M.C., et al.: Ranked Matching for Service Descriptions using OWL-S. In: *KiVS 2005. Kommunikation in verteilten Systemen*, pp. 91–102. Springer, Heidelberg (2005)
5. Jin, Y., Han, J.: Consistency and interoperability checking for component interaction rules. In: *Proc. of the 12th Asia-Pacific Software Engineering Conference*, pp. 595–602 (2005)
6. Mindswap: Mindswap OWL-S API. [Online]: <http://www.mindswap.org/2004/owl-s/api/>
7. Mokhtar, S.B., Georgantas, N., Issarny, V.: COCOA: Conversation-based service composition for pervasive computing environments. In: *ICPS 2006. Proc. of International Conference on Pervasive Services*, France, pp. 29–38 (2006)
8. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) *ISWC 2002. LNCS*, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
9. Pham, L.D., Colman, A., Schneider, J.-G.: Dynamic protocol aggregation and adaptation for service oriented computing. In: *ASWEC 2007. Proc. of the 18th Australian Software Engineering Conference*, pp. 39–48. IEEE Computer Society, Australia (2007)
10. Yu, J., et al.: Pattern based property specification and verification for service composition. In: Aberer, K., et al. (eds.) *WISE 2006. LNCS*, vol. 4255, pp. 156–168. Springer, Heidelberg (2006)