Web Service Protocols: Compatibility and Adaptation

Marlon Dumas
University of Tartu, Estonia
marlon.dumas@ut.ee

Boualem Benatallah, Hamid R. Motahari Nezhad
The University of New South Wales, Australia
{boualem,hamidm}@cse.unsw.edu.au

Abstract

This paper discusses the notion of protocol compatibility between Web services, and reviews a number of techniques for detecting incompatibilities and for synthesizing adapters for otherwise incompatible services. The paper also reviews related notions such as realizability, substitutability and controllability.

1 Introduction

The composition of Web services involves wiring together multiple web services and having them interact often in ways not originally foreseen during their initial development. In doing so, it is unavoidable that incompatibilities may arise and need to be identified and resolved. We classify these incompatibilities into two types: (i) *signature incompatibilities* that arise when a service requires an operation from another service, but this latter service does not offer it, or when a service A needs to exchange a message with another service B, but the schema of the message that A produces is not compatible with the one that B expects; and (ii) *protocol incompatibilities* that arise when a service A engages in a series of interactions with another service B, but the order in which service A undertakes these interactions is not compatible with that of B.

This paper discusses the notion of protocol compatibility between Web services and reviews a number of techniques for detecting incompatibilities and for synthesizing adapters for otherwise incompatible services. The paper also reviews related notions such as realizability [6], substitutability [4] and controllability [10].

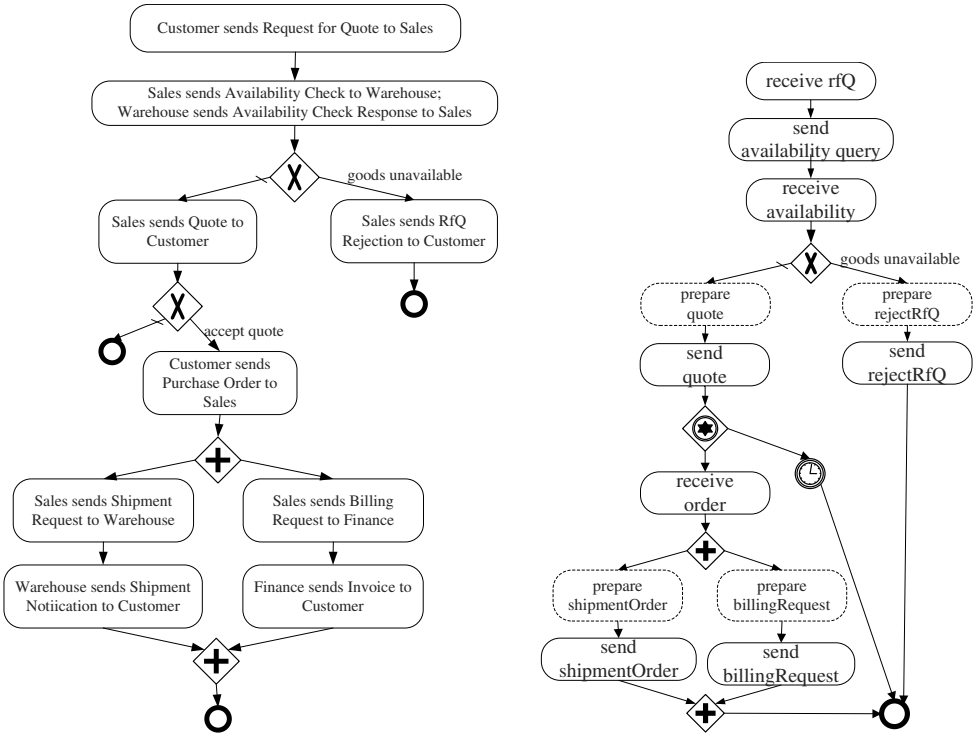
The next section introduces background concepts for modeling web service interactions in general, and service protocols in particular. Section 3 introduces the notion of protocol compatibility and related concepts. Section 4 discusses techniques for synthesizing adapters for protocol-incompatible services. Finally, Section 5 summarizes the discussion and raises directions for future work.

2 Service Interaction Modeling

It is customary to distinguish between two types of models of service interactions: choreographies and orchestrations [15]. A choreography describes interactions between a collection of services from a global perspective. In a choreography, no service plays a privileged role. Figure 1(a) depicts a choreography in the Business Process Modeling Notation (BPMN) [14]. Four services are involved in this choreography: customer, sales, warehouse and finance. Each activity denotes an interaction between two services. Importantly, a choreography only shows interactions, as opposed to actions performed internally by a service. In contrast, an orchestration describes the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

interactions between a designated service (the orchestrator) and a plurality of subordinated services. Figure 1(b) depicts an orchestration for the sales service. An orchestration may include internal actions or timeouts. For example, Figure 1(b) includes four actions internal to the sales service (the four "prepare" actions in dashed lines) and a timeout: After sending a quote, the sales service waits for an order until the quote's expiry time.



(a) Order management choreography (b) Orchestration for Sales service

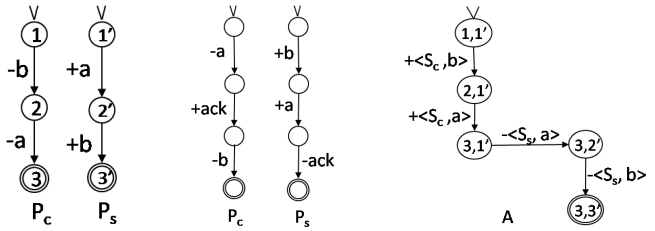
Figure 1: Examples: Choreography and Orchestration

If we consider a multi-party choreography and restrict it to those interactions that involve a given pair of services – e.g. the interactions between the sales and the customer services in the above example – we obtain a (bilateral) *service protocol*. Service protocols described from the perspective of one participant are also called *behavioral interfaces* [7], because they define the behaviour of a service vis-a-vis of one of its clients or peers.

The derivation of behavioural interfaces from choreographies may require refinements. Consider, e.g. the choice in Figure 1(a) that the customer performs between accepting or rejecting a quote. If the customer accepts the quote, it sends an order. Thus, when receiving an order the sales service knows that the customer accepted the quote. However, if the customer rejects the quote, it does not send any message. When deriving a behavioural interface for the sales service, one needs to insert either a timeout (as in Figure 1(b)) or an additional interaction through which the customer communicates the rejection to the sales service. Otherwise, the sales service will wait indefinitely for an order. The notion of *realizability* [6] (also called enforceability [18]) captures this issue. A choreography is *realizable* if the behavioural interfaces obtained by projection of the choreography into each of its participating *roles*, collectively enforce all control-flow constraints in the choreography.

Languages for specifying choreographies, protocols and orchestrations include BPMN (see above) and BPEL.[1] In BPEL, orchestrations are defined down to the point where they can be executed by dedicated platforms. Also, BPEL allows one to specify protocols/behavioral interfaces. For formal analysis, protocols may be represented using e.g. finite state machines (FSMs) [2], process algebra [11] or Petri nets [10, 3].

[1] `http://www.oasis-open.org/committees/wsbpel/`



(a) (b) (c)

Figure 2: Incompatibilities & adaptation: (a) unspecified reception, (b) deadlock, (c) adapter for protocols in (a)

3 Compatibility

Two services are protocol-compatible if every joint execution of these services leads to a proper final state, i.e. a state in which both services are in a final state in their respective protocols [2]. Under the assumption synchronous communication, Yellin & Ström [17] identify two main types of protocol mismatches: *unspecified reception*, in which one party sends a message while the other is not expecting it; and *deadlock*, the case where both parties are mutually waiting to receive some message from the other. To illustrate the concepts, consider the protocols of P_s (of service S_s) and P_c (of service S_c) in Figure 2(a): P_c sends message b (shown by a -b), while P_s does not expect to receive it (unspecified reception). In Figure 2(b) instead, P_c expects to receive message ack after sending a (shown by +ack), while P_s is waiting to receive b (+b). This is a deadlock case. Two protocols are said to be *compatible* if they have no unspecified receptions and they are deadlock-free.

The protocol A' obtained by reversing the polarity of every message in a protocol A is called the mirror of A. In other words, sent messages in A become received messages in its mirror A', while received messages become sent messages. In general, a service protocol is compatible with its mirror. However, if a protocol specification includes internal actions (e.g. timers or evaluation of boolean conditions resulting in certain branches being taken) it is possible that this protocol is not compatible with its mirror protocol, nor with any other protocol. If so, the service is said to be uncontrollable [10]. The problem of controllability is intuitively related to that of realizability – as that they both result when internal choices are not externalized as messages. However, a formal relation between controllability and realizability is yet to be established.

Replaceability (or *substitutability*) refers to the ability for a service to replace another one without inducing incompatibilities [4]. In ServiceMosaic [2], two main classes of replaceability are defined: *subsubmption* and *equivalence*. Protocol P_1 *subsumes* P_2 if P_1 supports at least all the execution traces that P_2 supports. If so, a service S_1 (with protocol P_1) can replace service S_2 (with protocol P_2). If P_1 subsumes P_2, and P_2 subsumes P_1, then P_1 and P_2 are equivalent, and services S_1 and S_2 can be used interchangeably. Finer notions of replaceability are defined in terms of bisimulation [3].

Finally, one can ask the question of whether an orchestration *conforms to* a protocol. If we take the orchestration and we project it to those interactions that appear in its protocol, the question is whether or not the projected orchestration is compatible with the service's protocol. This question is studied in [9].

4 Adaptation

When two services are incompatible, it may be possible to introduce an adapter to resolve their mismatches. In such cases, the service protocols are said to be *adaptable*. Depending on the types of mismatches, it may be possible to automatically synthesize an adapter. The question of synthesizing adapters for incompatible protocols has been studied in the area of SOA, as well as earlier in the area of component-based software engineering.

Yellin & Ström [17] propose an approach for checking the existence of an adapter for incompatible protocols. An adapter is modelled as an FSM consisting of a set of states, a set of typed memory cells to store the messages

received by the adapter, and a set of state transition rules. Each rule describes a transition from a state to another in the adapter based on sending or receiving messages, along with a set of memory actions that store or retrieve messages in/from the cells. A rule also constructs messages that need to be sent to partners. The adapter's protocol is said to be compatible with protocols P_1 and P_2 of the adapted components, if their interactions have no unspecified reception and are deadlock free. Figure 2(c) shows an example of an adapter for protocols in Figure 2(a). To synthesize the adapter specification for a pair of components, their interface mappings is required as the input (e.g. which messages should be mapped to which other messages). The adapter synthesis process explores all possible interactions between the protocols P_1 and P_2 and adds them to the adapter protocol. If there are states leading to deadlocks or with unspecified reception, they are removed from the adapter protocol.

Other proposals rely on alternative protocol specification languages that explicitly support concurrency. Mateescu et al. [11] propose a technique for adapter synthesis based on protocols specified using process algebra. Similarly, Brogi et al. [5] provide an automated adapter synthesis approach for protocols specified in BPEL.

Another line of research for service adapter development proposes to characterise the classes of possible mismatches between protocols, provides guidelines for users to identify them and proposes templates to resolve mistmatches based on design patterns [1] or composable adaptation operations [8]. In these approaches, the construction of adapters requires manual intervention. Some of these approaches, e.g. [8] deal with mismatch patterns not supported in automated approaches – e.g. mismatches where a message emitted by a service needs to be mapped to an unbounded number of messages in the receiving service.

Automated approaches for adapter generation make the following assumptions: (i) there is no mismatch at the interface-level, or the correct interface mappings have been provided as the input, and (ii) if there are interactions which lead to deadlocks, they are not adaptable. As discussed in [12], the interface-level mappings can not be always correctly identified without considering the protocol specifications. Second, some deadlock cases may be adaptable, e.g. the resolution of a deadlock may require the generation of messages (e.g., an acknowledgment) that can be constructed in the adapter via user-defined functions.

To address these limitations, Motahari Nezhad et al. [12] approach adapter development as an iterative process consisting of both interface-level and protocol-level mismatch identification and resolution. Their approach starts from an initial set of interface matchings, computed by matching the WSDL interfaces of services, and then, considering the protocol specifications of two services, identifies all the interactions that results in deadlocks. The result is presented in the form of a *mismatch tree* to the user, where the user can identify if such interactions are resolvable. The approach also helps the user by analyzing the mismatch tree. Some deadlock cases may be handled by going back to the interface matching step and refining the interface matchings.

5 Summary and outlook

Figure 3 summarizes the notions introduced in the paper. This panorama summarizes a significant body of research work in the area of service-oriented computing. In this body of work, research questions are often approached under the assumption that the choreographies, protocols and/or orchestrations are known and given as input. Sometimes however, these specifications are unavailable or they are incompletely or unreliably specified, yet one needs to make assertions regarding the correct behavior of a service-oriented system. Recent work has addressed the question of analyzing logs representing the observed behavior of a service-oriented system in order to determine if these logs conform to a choreography or protocol specification [16]. One of the key issues in this setting is that of "correlation", that is, how to group together log entries (such as those in message logs) to produce trails that represent conversations between two or more services [13]. Open questions in this area include investigating the application of techniques from machine learning and information clustering.

An open question in the field of service adaptation is how to maintain adapters in an environment where services evolve continuously. For example, given two services S1 and S2 that communicate through an adapter, how can this adapter be updated (with minimal effort) when either S1 or S2 evolve or are replaced?

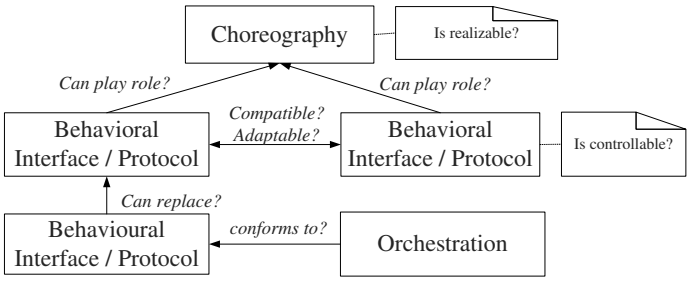


Figure 3: Relations between service interaction modeling viewpoints

References

[1] B. Benatallah, F. Casati, D. Grigori, H. Motahari Nezhad, and F. Toumani. Developing Adapters for Web Services Integration. In *Proc. of CAiSE*, pages 415–429, 2005.

[2] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3):327–357, 2006.

[3] F. Bonchi, A. Brogi, S. Corfini, and F. Gadducci. Compositional Specification of Web Services Via Behavioural Equivalence of Nets: A Case Study. In *Proc. of PETRI NETS*, pages 52-71, 2008.

[4] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? In *Proceedings of the 5th International Workshop on Technologies for E-Services (TES)*, pages 15–28, 2004.

[5] A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proc. of ICSOC*, 2006.

[6] T. Bultan, X. Fu, and J. Su. Analyzing conversations: Realizability, synchronizability, and verification. In L. Baresi and E. D. Nitto, editors, *Test and Analysis of Web Services*, pages 57–85. Springer, 2007.

[7] R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.

[8] M. Dumas, M. Spork, and K. Wang. Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In *Proc. of BPM*, pages 65–80, 2006.

[9] D. König, N. Lohmann, S. Moser, C. Stahl, and K. Wolf. Extending the compatibility notion for abstract WS-BPEL processes. In *Proc of WWW*, pages 785–794, May 2008.

[10] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data Knowl. Eng.*, 64(1):38–54, 2008.

[11] R. Mateescu, P. Poizat, and G. Salaün. Behavioral adaptation of component compositions based on process algebra encodings. In *Proc. of ASE*, pages 385–388, 2007.

[12] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proc. of WWW*, pages 993–1002, 2007.

[13] H. R. Motahari Nezhad, R. Saint-Paul, B. Benatallah, F. Casati, and P. Andritsos. Process spaceship: Discovering and exploring process views from event logs in data spaces. In *Proc. of VLDB*, 2008.

[14] Object Management Group. Business Process Modeling Notation, V1.1. *OMG Available Specification*, January 2008.

[15] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.

[16] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance checking of service behavior. *ACM Trans. Internet Techn.*, 8(3), 2008.

[17] D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):292–333, 1997.

[18] J. M. Zaha, M. Dumas, A. H. M. ter Hofstede, A. P. Barros, and G. Decker. Service interaction modeling: Bridging global and local views. In *Proc. of EDOC*, 2006.