On Temporal Abstractions of Web Service Protocols

Boualem Benatallah[1], Fabio Casati[2], Julien Ponge[3], and Farouk Toumani[3]

[1] CSE, UNSW, Sydney NSW 2052, Australia (boualem@cse.unsw.edu.au)
[2] Hewlett-Packard Laboratories, Palo Alto, CA, 94304 USA (casati@hpl.hp.com)
[3] LIMOS, UBP Clermont-Ferrand, France ({ponge,ftoumani}@isima.fr)

1 Introduction

Web services are increasingly gaining acceptance as a framework for facilitating application-to-application interactions within and across enterprises. They provide abstractions and technologies for exposing enterprise applications as services and make them accessible programmatically through standardized interfaces. However, tools supporting service development today provide little support for high level modeling and analysis of abstractions at higher level of services stack, and in particular there is little support for protocol modeling and management. We believe that indeed protocol modeling and management will be key in supporting Web service development and interaction, and that developing formal models and a protocol algebra will have a positive impact similar to the one that the relational model and the relational algebra had in database technology.

When developing our framework for service protocols modeling, analysis, and management [1, 2], we identified the need for representing temporal abstractions in protocol descriptions. In particular, our analysis of the characteristics and requirements of service protocols in terms of description languages, we found that, in addition to message choreography constraints, protocol specification languages need to cater for time-sensitive conversations (i.e., conversations that are characterized by temporal constraints on when an operation must or can be invoked). For example, a protocol may specify that a `purchase order` message is accepted only if it is received within 24 hours after a `quotation` has been received. In this paper, we discuss the augmentation of business protocols with specifications of temporal abstractions (called timed protocols). Then we motivate, through examples, a need for analyzing timed protocol specifications, and specifically for identifying if and under what conditions two services, characterized by certain timed protocols, can interact. Technical details are given in an extended version of this paper [3] where a formal timed business protocol model is presented and operators that enable characterizing compatibility and replaceability classes for timed protocols are described.

2 Modeling temporal abstractions in business protocols

In our approach, business protocols are modeled as deterministic finite state machines, where the states represent the different phases in which a service may

Proceedings of the CAiSE'05 Forum - O. Belo, J. Eder, J. Falcão e Cunha, O. Pastor (Eds.)

go through during its interaction with a requestor. Transitions are triggered by messages sent by the requestor to the provider or vice versa (hence, transitions are labeled with either input or output messages). As an example, Figure 1 shows a graphical representation of a protocol P that describes the external behavior of an order management service that allows users to buy some kinds of goods. Each transition is labeled with a message name followed by the message polarity, that is, whether the message is incoming (plus sign) or outgoing (minus sign) [4][4].

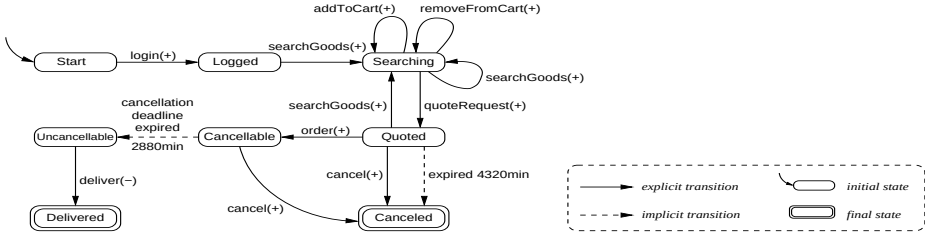

Fig. 1. A sample timed business protocol P.

In our previous work on protocol modeling [1], we identified that catering for temporal abstractions in protocol descriptions is an important requirement. In particular, our analysis of the characteristics and requirements of service protocols in terms of description languages, we found that, although most state transitions occur due to explicit operation invocations, there are cases in which transitions occur without an explicit invocation by requesters. We refer to these transitions as *implicit transitions*. The large majority of implicit transitions are due to timing issues (deadline expirations). For example, many services allow requestors to reserve a resource or to perform certain actions (invoke certain operations) only within a time window, after which these operations cannot be performed any more. For example, consider again the protocol P of Figure 1. This protocol specifies that when the service enters the state Quoted, a quotation is valid only for 3 days (equal to 4320 minutes), a time interval within which the user can order the selected goods (operation order). After this period of time, the conversation moves to the final state canceled, denoting that the server has canceled the order (implicit transition *expired* with a temporal constraint 4320min). Note that the implicit transition *expired* imposes time constraints on all transitions that can be fired from state Quoted (i.e., the operations order, searchGoods and cancel), as once it fires it leads the conversation to a state from which those operations are not allowed. An analogous reasoning can be applied to transition Cancellation deadline expired.

We use the term *timed busisness protocol* (or timed protocol for short) to denote a business protocol whose definition contains timed transitions. The se-

[4] In this paper, we use the notation $m(+)$ (respectively, $m(-)$) to denote that m is an input (respectively, output) message.

mantics of timed protocols is based on the notion of timed traces. For example, consider an execution of a service S that supports a protocol P of figure 1. We use the expression (searchGoods(+),1) to denote the occurence of the message searchGoods at an instant t=1. t represents the elapsed time since the beginning of the execution of S. A *timed trace* of a protocol is then defined as a sequence of such pairs. As an example, the sequence of pairs (login(+),0) . (searchGoods(+),1) . (addToCart(+),3) . (quoteRequest(+),7) . (cancel(+),120)) is a timed trace which is compliant with the protocol P.

3 Compatibility and replaceability in timed protocols

This section discusses the problem and motivates the need for timed protocol analysis (and specifically compatibility and replaceability analysis). Once services are endowed with protocol specifications, protocol management operators can be identified to perform the following type of analysis: (1) *compatibility analysis* refers to checking if the protocol of a requestor and of a provider are compatible, that is, if conversations can take place between the services, and (2)*replaceability analysis* refers to checking if a service provider R can replace another service provider S from a protocol standpoint, that is, if R can support the same conversation that S supports.

For both compatibility and replaceability, we have defined *classes* to identify different levels of compatibility and replaceability, as well as *operators* that can be applied to protocol definition to asses the level of compatibility and replaceability [2]. In the sequel, we discuss the novel opportunities and needs that timed protocols bring in this regard.

Compatibility in timed protocols We present several examples related to protocol compatibility (resp., replaceability) analysis, starting from a simple case to more complex ones. Consider protocol P depicted on Figure 1 and its reversed protocol P' obtained from P by reversing the polarity of the messages (i.e., input messages becomes outputs and vice versa). P' can interact correctly with P in the sense that, considering a given interaction between these two protocols, whenever P' sends a message at an instant t, the protocol P could receive it and vice versa. For example, protocol P' supports the following complete timed trace: (login(-),0) . (searchGoods(-),1) . (addToCart(-),2) . (quoteRequest(-),3) . (cancel(-),4) In this trace, cancel is the only operation whose temporal availability is restricted since both P and P' have an implicit transition that is fired 4320 minutes after having entered the Quoted state. In the previous trace the cancel message is sent by P' only 1 minute after the quotation has been performed, hence the message is legal.

We now illustrate a simple example of incompatibility between two protocols. Consider a protocol P' that supports the following timed trace: (login(-),0) . (searchGoods(-),1) . (addToCart(-),2) . (quoteRequest(-),3) . (cancel(-),4350). During such an execution, P' cannot interact correctly with the protocol P of

Figure 1. Indeed, P' will fire the operation cancel 7347 minutes after the quotation has been performed, which is more than the 4320 minutes allowed by P (i.e., P has already moved to the Canceled state).

The previous cases were simple to check because it was sufficient to compare pairs of states locally. The following example illustrates a more complex case. Consider the protocols P and P' depicted on Figure 2. Unlike the previous exam-

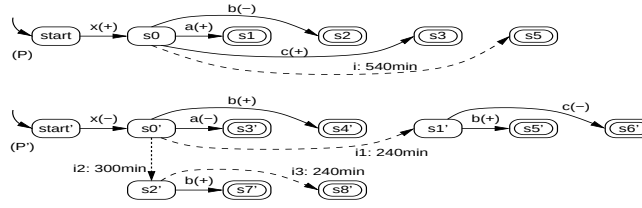

Fig. 2. Two compatible timed protocols.

ples, the two protocols have very different shapes. For instance, we can observe that after the execution of the operation x the protocols P and P' move, respectively, to the states s_0 and s'_0. These states do not offer the same operations (at least if we consider the operations that are defined explicitly at these two states). The state s_0 provides the operations a, b and c while the state s'_0 only provides the operations a and b. Consequently, focusing compatibility checking only on these two states is not enough. Indeed the operation c for example may be available for a client interacting with this service after 240 minutes. This is due to the presence of an implicit transition i_1 that automatically leads a service to the state s'_0 from which c can be fired. Note that to check if protocol P and P' are compatible we need also to consider all the states that are automatically (implicitly) reachable from a given state. In our case, checking if s_0 and s'_0 are compatible implies that we also consider s'_1 and s'_2 since they can be reached from s'_0 through i_1 and i_2. More precisely, we need to make explicit all the operations, and their associated timing constraints, that are available at these states. For example, as given below, looking to the implicit transitions, we can derive the temporal availabilities of the operations at the states s'_0 and s_0.

$$(\mathsf{P}) \begin{cases} \mathsf{a} : [0min, 540min] \\ \mathsf{b} : [0min, 540min] \\ \mathsf{c} : [0min, 540min] \end{cases} \qquad (\mathsf{P}') \begin{cases} \mathsf{a} : [0min, 240min] \\ \mathsf{b} : [0min, 780min] \\ \mathsf{c} : [240min, 540min] \end{cases}$$

Operation a will be performed by P' during a temporal window where P is ready to accept the message fired by P'. The same is true for c. The case of b is different since it is P that sends the message. The temporal window defined by P' for receiving the related message is wider than the one used for P to send it, thus P' is ready to receive a message b fired by P. We see that conversations can take place between P and P' as the messages can be exchanged during the allowed temporal slices defined by each protocol. However the compatibility between s_0

and s'_0 is not obvious since several other states have to be taken into account to get to the conclusion that a compatibility is effectively possible. However, note that compatibility is dependent on timing. In fact, not all conversations that can be generated by the client (P) can be supported by the provider (P'). If the client implementation is such that the client sends a message c right away after a message x, then it will cause the provider to respond with a fault message.

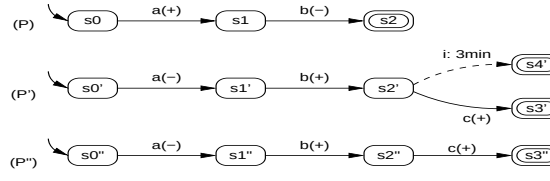

Fig. 3. Another compatibility problem illustrated.

Finally, the following example shows that implicit transitions can also influence the identification of final states and this naturally impacts compatibility analysis. Let's consider the 3 protocols P, P' and P'' depicted on Figure 3. We can observe that when interacting with P' or P'', the protocol P will reach its final state after executing the operations a and b while P' and P'' both remain at intermediary states (respectively, the states s'_2 and s''_2). Clearly, this is not a problem for P' as this protocol is able to automatically reach a final state from the state s'_2, and hence, its terminates correctly the conversation. Therefore, the interaction of P with P' is correct. However, P and P'' are not compatible since P'' remains in an intermediary state and will not be able to terminate correctly its execution (i.e., to reach a final state).

The above discussion has emphasized the need for a new compatibility class, called *time-dependent compatibility*. Protocols P and P' have time-dependent compatibility if they are compatible only when they exchange messages following certain time constraints. Hence, time-dependent compatibility is a kind of partial compatibility. Note that an implementation of a client may be able to read the service provider's protocol and time its interaction so that messages are sent when allowed. The discussion of such "adaptive" implementations is outside the scope of this paper, since as mentioned here we limit to protocol analysis without discussing service implementation and compliance.

Correspondingly, the discussion has also emphasized the need for operators that identify these time constraints resulting from the joint compatibility analysis of the two protocols, as shown earlier.

Replaceability in timed protocols We now turn our attention to the replaceability problem. We will provide less examples here as the previous section has already given an indication of the issues that can arise.

Consider protocols P and P' depicted on Figure 4. Like previously, the two states s_0 and s'_0 do not offer directly the same operations as a is not available

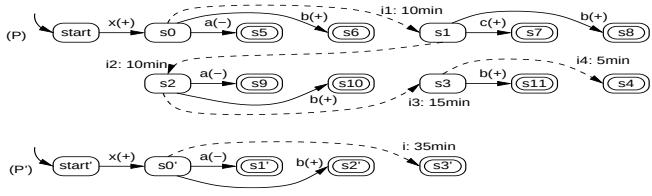

Fig. 4. A protocol P that can replace P'.

from s_0 while it is from s'_0 for instance. Again, let's have a look at the states that are implicitely reachable from s_0 and s'_0 to compute the temporal availabilities of the operations:

$$(P)\begin{cases} a: [0min, 10min], [20min, 35min] \\ b: [0min, 40min] \\ c: [10min, 20min] \end{cases} \qquad (P')\begin{cases} a: [0min, 35min] \\ b: [0min, 35min] \\ c: \emptyset \end{cases}$$

Protocol P can handle messages x, a and b from a client that is compatible with P' by looking at the temporal constraints. However P has an extra operation c. This operation being a message reception, it does not cause a problem. Indeed, a requestor or P' does not know about c since P' does not support it. Thus, they will never attempt to fire it. Finally, P can well replace P' by looking at s_0 and s'_0.

4 Discussion and Conclusions

In this paper, we build upon our earlier work on service protocols modeling, analysis, and management [1, 2] to cater for temporal abstractions in business protocols. In the extended version [3], we provide a formal characterization of compatibility and replaceability classes for timed business protocols as well as operators for analyzing these classes. This work is part of a larger framework supported by a CASE tool, partially implemented, that manages the entire service development lifecycle. The objective of the framework is to provide a comprehensive methodology and platform that can facilitate large-scale interoperation of Web services and substantially reduce the service development effort.

References

1. Benatallah, B., Casati, F., Toumani, F.: Web Service Conversation Modeling: A Cornerstone for e-Business Automation. IEEE Internet Computing **6** (2004)
2. Benatallah, B., Casati, F., Toumani, F.: Analysis and management of web services protocols. In: Procs of ER'04, Shanghai, China. (2004)
3. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On temporal abstractions of web services protocols (extended version). Technical report, `http://www.isima.fr/ponge/research.shtml#TR-BCPT05a` (2005)
4. Yellin, D., Storm, R.: Protocol Specifications and Component Adaptors. ACM Trans. Program. Lang. Syst. **19** (1997) 292–333