Interaction protocol mediation in web service composition

Liliana Ardissono*, Roberto Furnari, Giovanna Petrone and Marino Segnan

Dipartimento di Informatica,
Universitá di Torino,
Corso Svizzera 185,
10149 Torino, Italy
E-mail: liliana@di.unito.it
E-mail: furnari@di.unito.it
E-mail: giovanna@di.unito.it
E-mail: marino@di.unito.it
*Corresponding author

Abstract: This article presents a mediation framework supporting the integration of web services in orchestrated and choreographed services and the conciliation of interaction protocol mismatches. Our framework supports a loosely-coupled interaction among web services, based on the publish and subscribe pattern. Moreover, it manages web services as event-driven systems, in order to enable them to perform their operations in context-dependent way. By decoupling the web service interaction, our framework addresses several interaction protocol mismatches, ranging from differences in the signatures of the messages, to the order and number of the messages to be exchanged, including cross-protocol mismatches involving more than two peers.

Keywords: service-oriented computing; interaction protocol mediation; web service integration and synchronisation; application frameworks and architectures; web technologies.

Reference to this paper should be made as follows: Ardissono, L., Furnari, R., Petrone, G. and Segnan, M. (2010) 'Interaction protocol mediation in web service composition', *Int. J. Web Engineering and Technology*, Vol. 6, No. 1, pp.4–32.

Biographical notes: Liliana Ardissono is an Associate Professor at the Dipartimento di Informatica of the Università di Torino, where she obtained her university degree and her PhD in Computer Science. Her research interests include user modelling, adaptive hypermedia, service-oriented computing and cloud computing. She is the Secretary of the Board of Directors of User Modeling Inc. and she is a member of the Editorial Board of *User Modeling and User-Adapted Interaction – The Journal of Personalization Research*.

Roberto Furnari received a Laurea degree in Computer Science in 1999 at the Università di Torino. In 2010, he received his PhD in Computer Science within the Doctoral School of Science and High Technology of the Università di Torino. His main research interests are in web service orchestrations and choreographies.

Giovanna Petrone is a Researcher of Computer Science at the Università di Torino. Her research interests concern two main areas: multi-agent systems (with specific interest for distributed systems and web services), intelligent user interfaces (with specific attention to personalisation in web-based services) and cloud computing. Previously, she has worked for several years as a Software Engineer and Architect in large US and Italian computer companies and she was also a Visiting Scholar at the Stanford University.

Marino Segnan is a Researcher at the Computer Science Department, Università di Torino, working with the Advanced Service Architectures group. His recent research activities deal with interaction models for web services, choreographies, monitoring and cloud computing. His previous activity focused on the development of a qualitative simulation tool for model-based diagnosis. Previously, he worked with several companies, mainly on projects involving integrated development environments, user interfaces, compilers and transaction management.

1 Introduction

In service-oriented computing (Papazoglou et al., 2003, 2006), orchestrations and choreographies are introduced to specify composite applications whose business logic is managed, respectively, in a centralised or decentralised way (see Peltz, 2003). Specifically, in an orchestrated service, a centralised application controls the business logic of the composite service and invokes the web service suppliers. Differently, a choreographed service results from the decentralised cooperation of the participant web services.

The set-up and management of a composite service is challenged by the heterogeneity of the services to be integrated: although, service-oriented computing abstracts from the details of the execution platforms, it does not address protocol mismatches, which are solved by developing ad hoc adapters. In a composite service, this limitation makes web service replacement a hard task because it requires developing a different adapter for each alternative service. Moreover, it challenges the individual web service supplier by imposing the development of an adapter for each target composite service [e.g., see Ortiz et al. (2006) for a discussion on this topic].

In order to address such issues, we propose a mediation framework supporting the conciliation of interaction protocol mismatches. Our work builds on the idea that a general model for composite service management can be developed by abstracting from the flow details imposed by message-oriented coordination. Indeed, during the execution of a composite service, several message flow details can be ignored, provided that the data dependencies and synchronisation constraints imposed by the business logics of all the involved web services are respected. Our framework is thus based on:

- a declarative description of the synchronisation constraints which the web service participants have to respect, and of the mappings to be applied in order to conciliate the mismatches
- the runtime management of web services as event-driven systems coordinated in a loosely-coupled way, by means of a shared context.

We have assessed the flexibility and applicability of our framework to real world cases by exploiting it in the development of an orchestrated and of a choreographed e-commerce application. In this article, we describe the latter one.

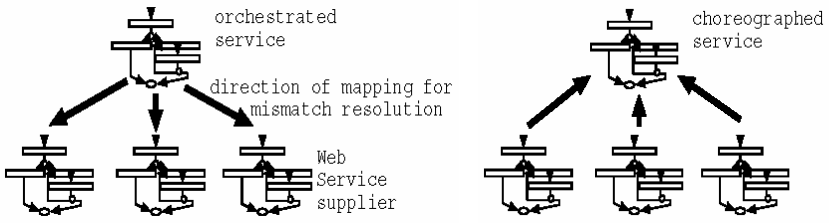
The remainder of this article is organised as follows: Section 2 describes the interaction protocol mismatches we address. Section 3 and Section 4 present the framework architecture and the proposed mediation model. Section 5 describes the management of composite services. Section 6 presents our approach to mediation. Section 7 provides some technical details and Section 8 describes a use case we have developed to validate our mediation model. Section 9 provides some validation results. Section 10 describes the related work and Section 11 concludes the article.

2 Protocol mismatches

When setting up a composite service, individual web services must be integrated and protocol mismatches have to be dealt with. Depending on the fact that the composite service is orchestrated or choreographed, there are two points of view:

- If the orchestrated service has to invoke a set of web services 'as they are' (e.g., widely used services), the interaction protocol of the orchestrator has to be mapped to the interaction protocols of the web services to be invoked; see Figure 1(a). Typically, alternative suppliers, having different interaction protocols, might be exploited to fill a certain role; therefore, a separate mapping has to be done for each service provider in order to support their replacement.
- In the case of a choreographed service, the interaction protocols of the participant web services have to be mapped to the choreography, which describes the contract to be respected by all the partners; see Figure 1(b).[1]

Figure 1 (a) Addressing mismatches in an orchestrated service (b) addressing mismatches in a choreographed service



(a) (b)

Note: In both cases, the arrows indicate the direction of mapping for mismatch resolution.

In service-oriented computing, the business logic of a composite service is specified as a workflow including the invocations of the Web Service Description Language (WSDL by W3C, 2001) operations which the participant web services have to perform. This specificity generates several mismatches, because the business protocols adopted by the web services are collapsed with their interaction protocols. Therefore, although some web services might collaborate at the business level, their cooperation is obstacle at the interaction level (see Williams et al., 2006).

Various types of mismatch challenge the web service interaction. Although such problems concern the interaction between web service consumers and providers in general, we discuss them in the context of choreographed and orchestrated services. Notice that we focus on protocol mismatches, leaving data mismatches apart, as they deserve separate treatment (e.g., see DERI International, 2005, 2008; Mrissa et al., 2007).

Table 1 shows a list of protocol mismatches. Although, the list is not exhaustive, it describes fairly frequent situations, which were also analysed in previous work on service adaptation (e.g., see Benatallah et al., 2005).

- Row 1 shows a mismatch in the signatures of the message respectively sent and received by the web services. The mismatch concerns the message name and the order of the parameters.
- In row 2, an outbound message corresponds to two or more inbound messages: web service B expects to receive the business data in different interaction steps, but web service A produces the data items in a single step. This mismatch is denoted as *message merge* in Benatallah et al. (2005).
- Row 3 is the dual of row 2 and shows a *message split* mismatch (Benatallah et al., 2005).
- In row 4, the recipient of the message does not expect some data items produced by the sender and should ignore the extra-information.
- In row 5, a web service sends a message which the receiver does not expect (*missing message*). The receiver should ignore the message.
- Row 6 shows the dual case: web service B expects a message which web service A does not send (*extra message*). This is a starvation case.
- Row 7 shows a typical deadlock situation involving web services A and B.

Table 1 Protocol mismatch examples

#	*Web service A*	*Web service B*
1	$\text{send}\left(\text{ms}_a\left(\text{x}_{i_1},...,\text{x}_{i_m}\right),\text{B}\right)$	$\text{receive}\left(\text{ms}_b\left(\text{x}_{j_1},...,\text{x}_{j_m}\right),\text{A}\right)$
2	send(ms(x_1, ...,x_n), B)	receive(ms$_1$(x_1, ..., x_i), A), ..., receive(ms$_r$(x_j, ..., x_n, A)
3	send(ms$_1$(x_1, ..., x_i), B), ..., send(ms$_r$(x_j, ..., x_n, B)	receive(ms(x_1, ..., x_n), A)
4	send(ms$_c$(x_1, ..., x_n), B)	receive(ms$_d$(x_1, ..., x_k, x_{k+l}, x_n), B)
5	send(ms(x_1, ..., x_n), B)	-
6	-	receive(ms(x_1, ..., x_n), A)
7	receive(ms$_e$(x_1, ..., x_n), B), send(ms$_f$(x_1, ..., x_n), B)	receive(ms$_f$(x_1, ..., x_n), A), send(ms$_e$(x_1, ..., x_n), A)

In addition to these well-known mismatches, which involve two services, we introduce the *wrong sender mismatch*, which can occur when more than two services interact with each other. For instance, the interaction protocol of a web service might specify that the web service expects to receive a message carrying a certain information item from a

certain peer. However, when the web service is integrated in a choreographed service, the choreography might specify that a different peer is going to generate the information item.

The protocol mismatches concerning the signatures of the WSDL operations, and those concerning the disalignments in the generation of acknowledgments (e.g., see rows 6 and 7 when the missing/extra message is an ack) can be solved by applying mappings easily managed by an ad hoc adapter. However, in order to solve other mismatches, the interaction among web services has to be deeply modified. Noticeably, the *missing*, *extra*, and *wrong sender* mismatches could be solved in a composite service by focusing on the business data needed by the participant web services and by abstracting from the identities of the services producing such business data.

In order to provide such capabilities, a flexible mediation framework is needed, which extends the protocol mismatch resolution from the one-to-one to the many-to-many case. Our work is aimed at answering this need.

3 The framework architecture

The research on distributed processes proposed several coordination models to enable the cooperation between heterogeneous systems, by abstracting from language and interaction protocol details (e.g., see Papadopoulos and Arbab, 1998). In particular, data-driven coordination models are based on the shared dataspace metaphor, which decouples processes in both space and time by modelling their interaction as the access to a common, content-addressable data structure; these models use tuple-based coordination languages, such as Linda (Ahuja et a., 1986) to post and retrieve data items from the shared dataspace.

We have adopted a shared dataspace model for managing the context of a composite service. Henceforth, such context is denoted as choreography context. Specifically, our framework is based on the architecture shown in Figure 2:

- Each web service is wrapped by an adapter, the communication interface (CI), which decouples its interaction with the rest of the composite service by inhibiting the direct invocation of WSDL operations on the other web services. The CI interacts with a CtxMg to receive the context information relevant for the web service execution and to propagate the information generated by the web service. On the basis of the available information (stored in a local context), the CI selects the WSDL operations to be performed and invokes them, managing the web service as an event-driven system (Lu and Sadiq, 2007). The CI KB is a knowledge base and specifies the synchronisation information needed to manage the web service in conformance with the business logic of the composite service.

- The context management service (CtxMg) manages the choreography context by storing the data provided by the web services via their CIs. When an information item is generated, the CtxMg propagates it to the interested CIs according to the publish and subscribe pattern, supporting distributed caching. The choreography context includes:

a basic coordination information; e.g., the identifier of the instance of the composite service to which the web services participate (see OMG Object Management Group, 2008)

b the business data of the composite service, to be shared by the participant web services

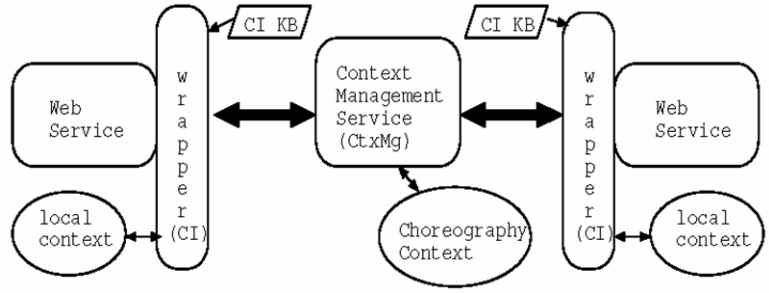
c the synchronisation information which the web services need to coordinate with each other.

In order to uniformly handle business data and synchronisation information, they are both represented as *tokens*.

The event-driven web service execution is achieved by subordinating the invocation of WSDL operations to the reception of enabling events. In our case, the events are the tokens associated to business data items and synchronisation information.

Notice that the choreography context is centralised in order to support the data consistency management, e.g., in the propagation of changes in the business data items. In contrast, the web service activities are managed in a distributed way in order to minimise the amount of service-dependent information which the CtxMg has to handle.

Figure 2 Framework architecture



4 Our mediation model

Interaction protocols have been traditionally represented by means of graph-based languages specifying the inbound and outbound messages to be exchanged by the services, and the partial order relations among such messages; e.g., see WSCI (Arkin et al., 2002). Moreover, more complex flow constructs have been introduced in Web Service composition languages in order to support the specification of typical workflow patterns occurring in business processes (e.g., see OASIS, 2005).

As our mediation model is aimed at supporting the interaction among web services in both web service orchestration and choreography, it takes web service composition languages as a reference for the selection of the flow constraints to be handled. Specifically, our model deals with the following patterns (see van der Aalst et al., 2008): sequence, parallel split, exclusive choice, synchronisation, simple merge and synchronising merge. The management of more complex patterns, such as cycles, is part of our future work.

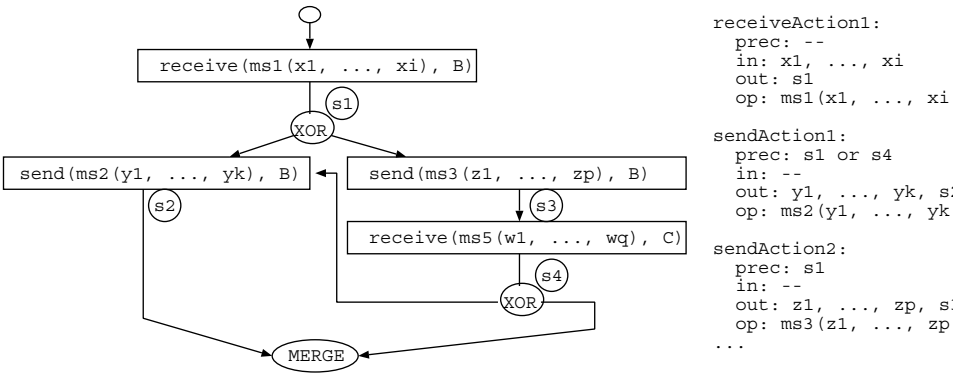
4.1 Token-based representation of an interaction protocol

In order to model the flow dependencies among messages in our event-driven approach, and to support a unified management of business and synchronisation information in the choreography context, we introduce the *token-based* representation of interaction protocols. This representation describes the dependencies among messages in terms of input and output tokens associated to the nodes of the corresponding interaction graph. An input token is needed to activate a node, while an output token represents the fact that the message described by the node has been processed (received or sent). Figure 3(a) shows a graph-based representation of an interaction protocol, decorated with tokens; in the graph, the inbound (outbound) messages are associated to `receive()` (`send()`) nodes.

In the token-based representation, each node of the interaction graph has an associated descriptor (`receiveAction` for inbound messages, `sendAction` for outbound ones) which specifies the following information:

- The `prec` field (precondition) specifies the synchronisation constraints on the management of the message imposed by the interaction protocol. The preconditions are Boolean conditions on synchronisation tokens and they enforce the partial order relations on messages, wherever their data dependencies are not enough to sort them. Specifically, the precondition of an inbound message defines the synchronisation constraints on the invocation of its object WSDL operation. Instead, the precondition of an outbound message specifies the synchronisation constraints on the generation of its output tokens.
- The `in` field specifies the input parameters of the WSDL operation which is the object of the message.
- The `out` field specifies the 'new' and the revised output data items (if any); moreover, it includes the synchronisation token(s) to be generated when the message is handled. The output data items propagated by the message without modifying them are not reported as they represent redundant information.
- The `op` field specifies the signature of the object WSDL operation.

Figure 3 (a) Interaction protocol of a web service, depicted as a graph decorated with synchronisation tokens (b) token-based representation of the interaction protocol



(a) (b)

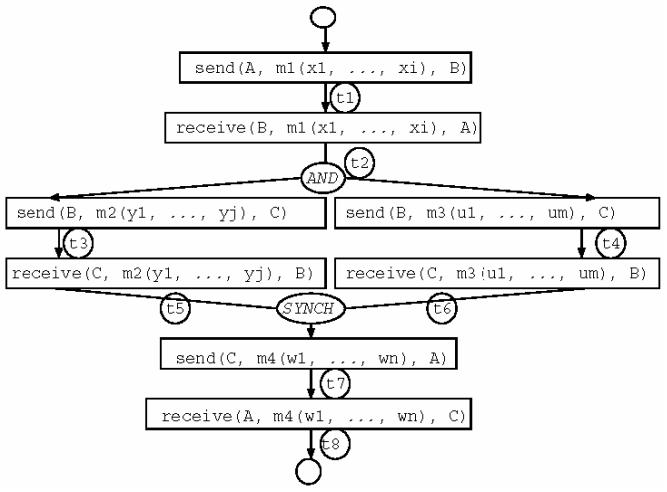
For instance, Figure 3(b) shows the token-based representation of the interaction protocol shown in Figure 3(a).

An interaction protocol can be automatically translated to its token-based representation by means of the following steps:

1 Each message must be associated with an output synchronisation token; see the s_i circles in Figure 3(a). The association means that, when the inbound (outbound) message is received (sent), the related synchronisation constraint is satisfied. Notice that an output arc may split to describe the parallel execution of paths in the interaction graph [AND split, corresponding to the *parallel split* pattern in van der Aalst et al. (2003)], the selection of one of a set of alternative paths [XOR, i.e., *exclusive choice* in van der Aalst et al. (2003)]. In these cases, we assumed that, at runtime, a single token is propagated to each ramification of the split.[2]

2 For each message, the related descriptor has to be generated. In particular, if the input arc of the node is simple, the precondition is the input token. If the input arc merges a set of parallel paths [SYNCH join, i.e., *synchronisation* in van der Aalst et al. (2003)], the precondition is the conjunction of the tokens placed on such arcs. Otherwise, the precondition is the disjunction of the tokens placed on such arcs (MERGE join, i.e., *simple merge and synchronising merge*).

Figure 4 Sample choreographed service (COMP-S), (a) described as an interaction graph (b) in the token-based representation



4.2 Token-based representation of a composite service

A composite service can be represented by means of an interaction graph similar to the one adopted for web services. If the service is choreographed, the graph represents a global view of the web service interaction. If the service is orchestrated, the graph represents the viewpoint of the orchestrator.

Figure 4(a) shows the interaction graph describing a sample choreographed service, henceforth, denoted as `COMP-S`. The graph includes a couple of nodes for each message

exchanged by the participants. The first node, `send()`, represents the act of sending the message and has three arguments: the message sender, the object message and the recipient. The other node, `receive()`, represents the act of receiving a message and specifies the recipient, the object message and its sender. The translation of the interaction graph to the token-based representation is similar to the one described in Section 4.1; Figure 4(b) shows a portion of the token-based representation of `COMP-S`.

4.3 Token-based representation of a web service within a composite service

The integration of a web service in a composite service requires a preliminary data and message mapping process, where the existing protocol mismatches are identified and solved. In this process, the token-based representation of the web service is specialised in order to take the global synchronisation constraints of the composite service into account. The revised token-based representation is stored in the CI KB of the web service, which is used at runtime in order to steer the web service execution. Before describing such process in detail in Section 6, we sketch it on a simple example.

Suppose that a web service, `WSA`, has to be integrated in `COMP-S` as a filler of role `A`. Figure 5 shows the choreography specification of `COMP-S` and the interaction protocol of `WSA`. In the figure, the mappings between local and target business data items are represented by using similar names for the parameters; e.g., `x1'` of `WSA` is mapped to `x1` in `COMP-S`. The dashed arrows show the mappings between the local and target messages: e.g., `mes1()` and `mes2()` of `WSA` correspond to `m1()` in the choreography; moreover, `mes3()` corresponds to `m4()`, but `WSA` expects to receive the message from the web service filling role `B(WSB)`; instead, in `COMP-S`, the sender is `C`; this is an example of a *wrong sender mismatch*. Given such correspondences, the interaction protocol of `WSA` can be mapped to the choreography by replacing the local synchronisation tokens with the target ones; e.g., local token `s2` must be replaced with target token `t1`.

Figure 5 Protocol mismatch resolution between web service `WSA` (at the right) and the composite web service `COMP-S` (at the left)

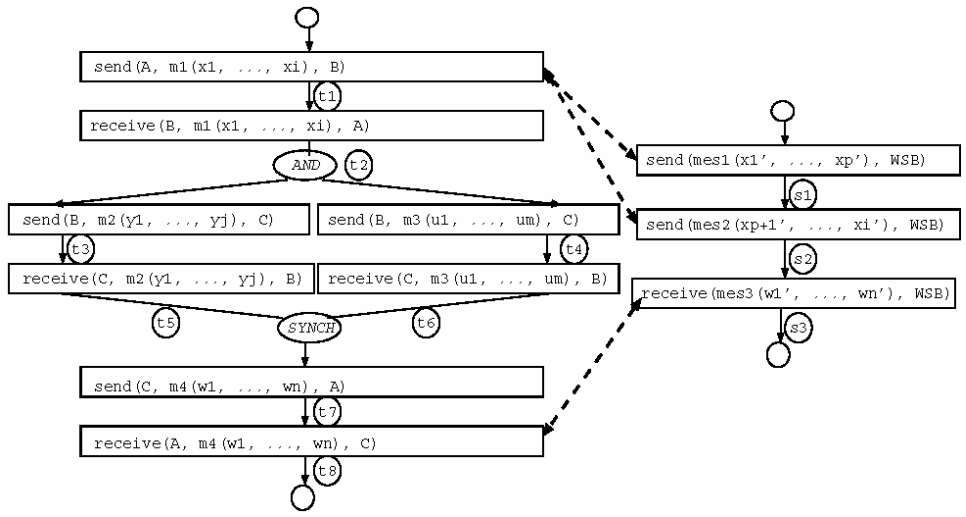
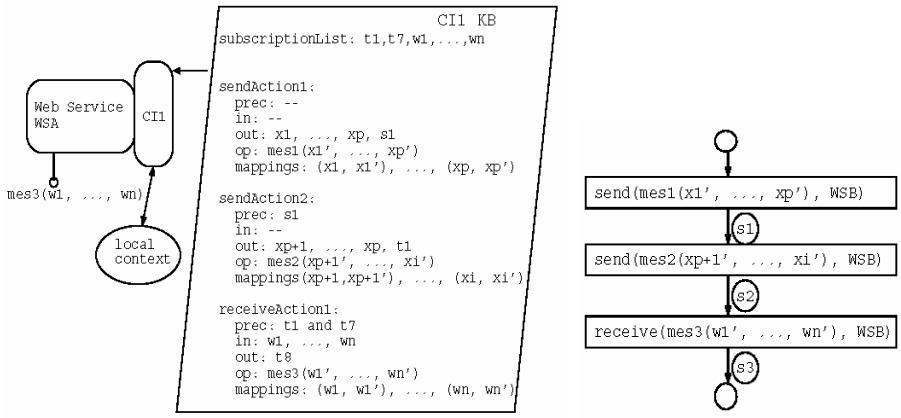


Figure 6(a) shows the CI KB of WSA; e.g., the receiveAction1 descriptor concerns the mes3() WSDL operation, which can be executed only after the synchronisation constraints represented by t1 and t7 have been satisfied. The arguments of the action (w1′, ..., wn′) are bound, respectively, to the w1, ..., wn business data items of COMP-S. Notice that the message descriptors in the CI KB have a mappings field which maps the business data items of the composite service to the local ones. Moreover, the CI KB has a subscriptionList slot reporting the business data items and the synchronisation tokens relevant for the web service execution.

Figure 6 (a) Action-based representation of the interaction protocol of WSA, within composite service COMP-S (b) interaction protocol of WSA



(a) (b)

5 Management of a composite service

The management of a composite service includes three phases: the *start-up phase*, the *initialisation phase* and the *runtime* phase.

The start-up phase is aimed at notifying the CtxMg about the references of the services which participate to a specific composite service. When a web service (wrapped by its CI) is launched, in order to play a role in a composite service, it pre-registers to the CtxMg, providing the required information in order to be contacted by the CtxMg later on.

When a composite service is invoked by a client, the *initialisation phase* is carried out in order to start a new interaction session. The CI of the service that starts the composite service sends the CtxMg a message asking for a new choreography session instance, according to the WS-coordination specifications (Oasis, 2009). In turn, the CtxMg creates the new session and replies with a unique choreography identifier. Moreover, the CtxMg sends messages to the other pre-registered services, in order to activate them on the new choreography instance. Notice that, although these activation messages conceptually belong to the initialisation phase, they can be sent in different moments during the composite service execution. For instance, in a book shopping scenario, a shipper service might be contacted only after a complex conversation between a service client and a service supplier has taken place.

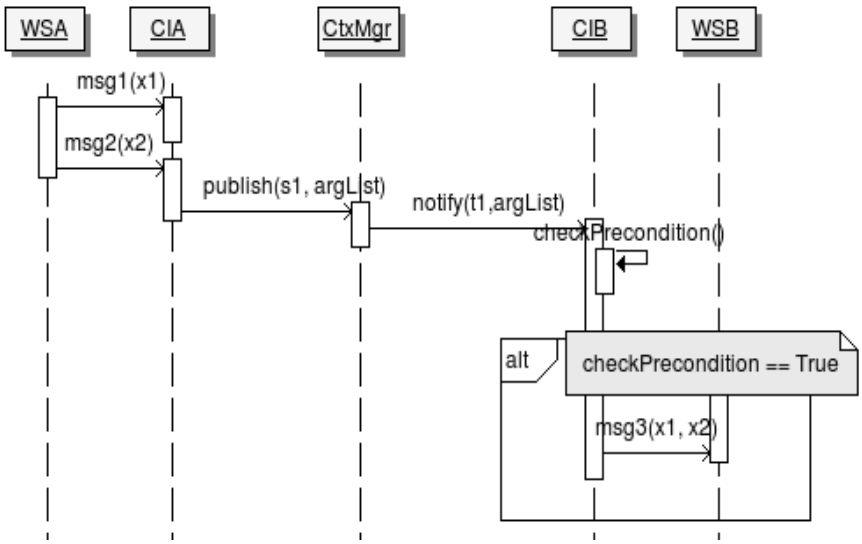
During the initialisation phase, the CI of each cooperating service also subscribes for the relevant context information (`subscriptionList` of the CI KB).

At *runtime*, the CIs manage the wrapped web services as event-driven systems acting in a shared context. Specifically, each CI mediates the interaction between a web service and the rest of the composite service by receiving the messages from the CtxMg and by intercepting the outbound messages of the web service. Moreover, the CI invokes the WSDL operations of the web service on the basis of the available context information.

- When a web service tries to invoke a WSDL operation on a supplier, the CI absorbs the outbound message, extracts the values of the output parameters and stores them in the local context. Then, the CI selects (by checking their preconditions and input data) the `sendAction` and `receiveAction` descriptors of the CI KB which can be performed, and executes them.
- When the CtxMg receives any data items or tokens, it stores them in the choreography context and propagates them to the CIs of the subscribed web services by sending a multicast `notify(arglist)` message.
- When a CI receives a set of data items and/or synchronisation tokens, it stores them in its local context. Then, it selects the enabled `receiveAction` and `sendAction` actions and performs them.

Figure 7 depicts a sample web service interaction. `WSA` tries to invoke operations `msg1` and `msg2` passing the `x1` and `x2` business data items. `CIA` (the CI of `WSA`) captures the message and publishes `x1`, `x2` and the related synchronisation tokens (`publish(arglist)`). In turn, CtxMgr notifies CIB. Finally, when receiving `x1` and `x2`, `CIB` invokes the `msg3(x1, x2)` operation on WSB.

Figure 7 Messages exchanged by web services at composite service runtime



6 Interaction protocol mediation

As discussed in Section 2, there are two different perspectives, depending on the fact that a web service has to be integrated in a composite service, or it has to invoke a set of web services, 'as they are'. In the following, we unify these perspectives by considering the local interaction protocol to be adapted and the target protocol to adapt to. The adaptation of the former to the latter is organised in three phases, listed in the following. In each phase, some unsolvable mismatches might be identified, in which case the integration fails. At the end of this process, the service developer can check the CI KB of the web service for correctness and completeness.

1 In the first phase, the parameters of the messages occurring in the local interaction protocol are mapped to the business data of the target protocol. As the identification of such correspondences is a complex task and requires a lot of domain knowledge, we assume that the service developer performs it. The output of this phase is the *data mapping table*; e.g., see Table 2.

   Notice that the target interaction protocol might require some business data item which the local protocol does not provide, or vice versa. If the data item is an acknowledgment, the item can be mapped to a default value in the *data mapping table*; otherwise, the mapping fails. Conversely, if the local protocol produces a data item which is not required in the target one, the item can be mapped to itself.

Table 2 Data mapping table of web service WSA within composite service `COMP-S`

#	*WSA business data*	*Target business d*
1	x1'	x1
2	x2'	x2
...	...	...
$	wn'	wn

2 In the second phase, the messages of the local interaction protocol are mapped to those of the target protocol; for the same reasons expressed above, we assume that the service developer carries out this task. During this phase, the developer might identify some local messages which have no correspondence in the target protocol, or vice versa; in such cases (s)he has to decide whether such conflicts can be solved, or they are fatal and they make the mapping fail. The mapping of messages is carried out as follows:

   - if the web service has to adapt to the interaction protocol of a supplier, the outbound (inbound) messages of the web service have to be mapped to the inbound (outbound) messages of the supplier
   - if the web service has to be integrated in a choreographed service, playing role *R*, the inbound (outbound) messages of the web service must be mapped to the inbound (outbound) messages of the filler of role *R*.

   This task produces the *message mapping table*, which relates the messages of the local protocol to the target messages; e.g., see Table 3. Two things are worth mentioning: first, the table may have incomplete rows, corresponding to the

unmapped messages. Second, if a web service is invoked more than once (e.g., in alternative paths of the composite service), the table contains duplicated rows because each path has to be separately mapped.

Table 3 Message mapping table of web service WSA within composite service `COMP-S`

#	*WSA messages*	*Target messages*
1	send(mes1(x1', ..., xp'), WSB)	send(A, m1(x1, ..., xi), B)
	send(mes2(xp + 1', ..., xi'), WSB)	
2	receive(mes3(w1', ..., wn'), WSB)	receive(A, m4(w1, ..., wn), C)

3 Starting from the data and *message mapping tables*, an automated procedure maps the tokens defined in the token-based representation of the local protocol to the tokens of the target protocol. This task produces the *token mapping table*; e.g., see Table 4. Then, given the three mapping tables, and the token-based representations of the local and target protocols, the same procedure generates the CI KB of the web service.

Table 4 Token mapping table of web service WSA within composite service `COMP-S`

#	*WSA tokens*	*Target tokens*
1	s1	s1
2	s2	t1
3	s3	t8

The generation of the CI KB consists of specifying the `receiveAction/sendAction` descriptors and the `subscriptionList`. For each row of the *message mapping table*, a `receiveAction/sendAction` descriptor is defined, depending on the fact that the message is an inbound or an outbound one for the service to be adapted:

- The `op` field must include the signature of the WSDL operation, as reported in the token-based representation of the local interaction protocol. If the protocol does not contain the message (*extra message* mismatch), the op field includes the signature of the operation specified in the target message.
- The `prec` field is defined by applying the mapping rules defined in Section 6.1 Specifically, the Boolean condition concerns the global and local synchronisation constraints affecting the message, represented ad conjunctions/disjunctions of target tokens.
- The `in` field must include the target business data items which, in the *data mapping table*, correspond to the input arguments of the WSDL operation (if any).
- The `out` field is defined by applying the mapping rules defined in Section 6.1. This field must include the target business data items which, in the *data mapping table*, correspond to the local output data of the WSDL operation. Moreover, it must include the target tokens which, in the *token mapping table*, correspond to the output tokens of the message.

- The `mappings` field reports the mappings of the *data mapping table* concerning the WSDL operation.

Given the descriptors, the `subscriptionList` is derived by including all the business data items and the synchronisation tokens occurring in their `prec` or `in` fields.

6.1 Rules for mapping interaction protocols

The mapping rules determine, for each message, the precondition (`prec` field) and the output information (out field) to be generated. These rules are applied to each row of the *message mapping table*, in order to generate a descriptor for each message occurring in the local or in the target interaction protocol. However, the order of application depends on the local protocol: starting from its start node, and following the paths of the interaction graph, the rules are applied in order to map the local messages to the target ones. Then, the messages of the target interaction protocol which are not mapped to any local messages are considered.

In the following, we report some of the rules. We denote the message(s) belonging to the local interaction protocol of the web service as *LM*, LM_i. Moreover, we denote the messages belonging to the target interaction protocol as *TM*, TM_j.

- *One-to-zero mapping*. Suppose that *LM* cannot be mapped to any message of the target interaction protocol (*missing message* mismatch). If *LM* is an outbound message, it can be ignored in the composite service. If it is an inbound one, but it does not represent an unsolvable conflict, it can be handled by setting by default the expected business data items. Specifically:
  a `prec`: let `COND` be the precondition of *LM* in the token-based representation of the local interaction protocol. Moreover, let's assume that the parent nodes of *LM* in the local protocol have been mapped to the target messages. Then, the precondition of *LM* in the CI KB is derived from `COND` by replacing the local tokens with the target ones in the *data mapping table*. Intuitively, *LM* has to be synchronised locally to the web service, by exploiting the tokens available in the context of the composite service.
  b `out`: the output business data, if any, is the set of target business data items associated to the local output data of *LM* in the *data mapping table*. The output token s of *LM* is maintained in the composite service to enforce the local synchronisation of messages within the web service.
- *Zero-to-one mapping*. Suppose that a target message TM cannot be mapped to any message of the local interaction protocol (*extra message*). In this case, the CI KB must include a `sendAction` descriptor for *TM*, which propagates to the coordination context the expected business data items and output token:
  a `prec`: the precondition is the one of *TM* in the target protocol
  b `out`: the output token is the one of *TM*.
- *One-to-one mapping*. Suppose that *LM* is mapped to message *TM*, as shown in Figure 8(a). Then:

a `prec`: *LM* gets the precondition of *TM*. Moreover, the local synchronisation in the web service must be enforced. For this purpose, if in the local interaction protocol *LM* directly follows one or more messages which are not mapped to any target message, the precondition must be joint with the output tokens of such messages, depending on the kind of arc connecting such messages to *LM* (simple arc, SYNCH, MERGE).[3]

b `out`: the output business data is the set of (target) business data items associated to the local output data of *LM* in the *data mapping table*. The output token s of *LM* is replaced with the output token t of *TM*.

- *One-to-n mapping*. Suppose that *LM* is mapped to messages TM_1, ..., TM_n, as shown in Figure 8(b). This means the contribution of *LM* corresponds to the joint contribution of all the target messages. Suppose also that TM_1, ..., TM_n are located in a single path of the interaction graph, or on different paths, joint by means of a SYNCH arc. Then:

  a `prec`: the precondition of *LM* includes the conjunction of the preconditions of TM_1, ..., TM_n, cleaned from the conjuncts corresponding to the output tokens of such messages. Specifically:

    1 If there are no order relations among messages in TM_1, ..., TM_n, then the tokens occurring in their preconditions are distinct from their output tokens. Thus, the precondition of *LM* must include the conjunction of the preconditions of TM_1, ..., TM_n.

    2 If there are some order relations in TM_1, ..., TM_n, for each order dependence $TM_h \rightarrow TM_k$, the preconditions of TM_k include the output tokens of TM_h. However, *LM* replaces both messages; therefore, such order dependence must be ignored when handling *LM*. Overall, this is achieved by removing the output tokens of TM_1, ..., TM_n from the precondition of *LM*.

  Finally, as for the previous cases, if in the local interaction protocol *LM* directly follows some local messages which are not mapped to any target message, the precondition must be joint with the output tokens of such messages, depending on the kind of arc entering such messages to *LM*; see footnote 3.

    1 `out`: the output business data is as above. The output token s of *LM* must be replaced with the output tokens t_1, ..., t_n of TM_1, ..., TM_n.

- *N-to-one mapping*. Suppose that LM_1, ..., LM_m are mapped to message TM, as shown in Figure 9. This means that the joint contribution of LM_1, ..., LM_m corresponds to the contribution of *TM* (message split). Suppose also that LM_1, ... LM_m are located in the same path of the local interaction protocol, or on different paths, which join in a SYNCH node. Then:

  a `prec`: the precondition of each LM_i message must be the precondition of *TM*. Moreover, it must include the local tokens needed to synchronise LM_i with respect to LM_1, ..., LM_m and the existing unmapped local messages (see footnote 3).

  b `out`: the output tokens are computed as usual.

Figure 8 (a) One-to-one mapping between a message of the local interaction protocol and a target interaction protocol (b) one-to-*n* mapping (see online version for colours)

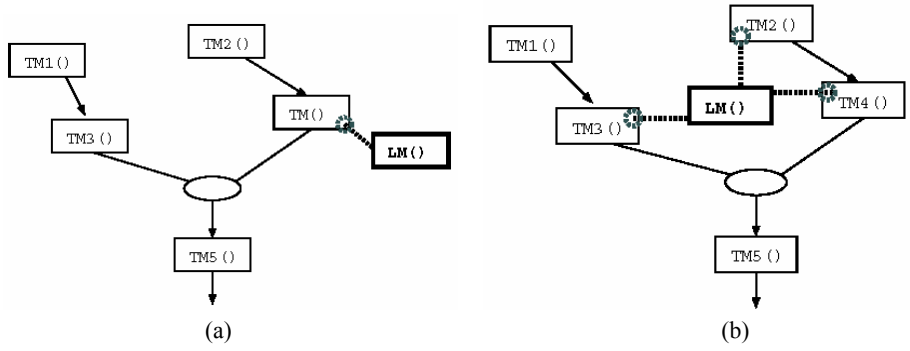
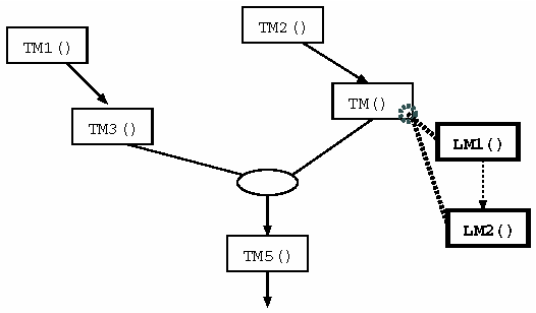


Figure 9 *N*-to-one mapping between a message of the local interaction protocol and a target interaction protocol (see online version for colours)



7 Technical details

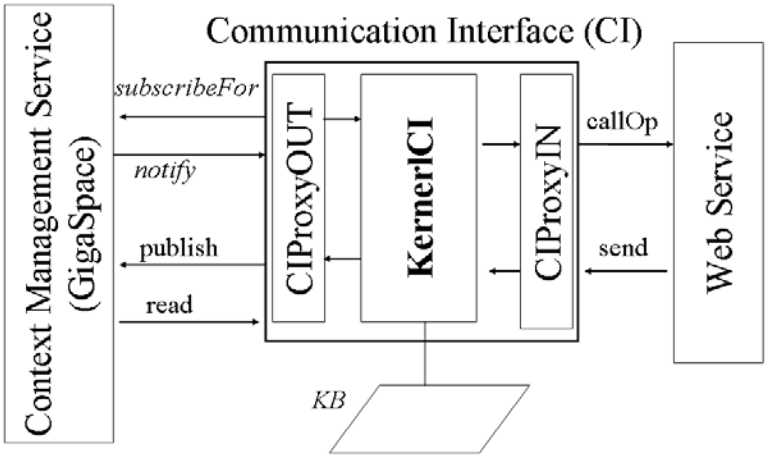
The prototype we developed as a reference implementation of our mediation framework has two main components: the tool implementing the CtxMg and the CI wrapping the individual web service.

The CtxMg has been developed by exploiting the GigaSpaces (2008) middleware, which offers a robust implementation of the JavaSpaces (JINI, 2008) paradigm supporting the propagation of data items according to the Linda (Ahuja et al., 1986) tuple space model. GigaSpaces provide a scalable environment where clients can publish and read information. Moreover, it offers a publish and subscribe mechanism that enables clients to subscribe for notification on events (e.g., creation, update, deletion) concerning particular data items. GigaSpaces also provide utilities and facilities enabling a Java-based application to connect to the tuple space in an easy and transparent way. Furthermore, it supports a secure transmission of information based on authentication and encryption.

Figure 10 shows the software architecture of our prototype. In particular, the CI consists of three main components:

- the *KernelCI* offers the core CI functionalities and implements the web service execution model presented in Section 5
- the *CIProxyOut* interface mediates the interaction with the CtxMg and implements the API needed to:
  a publish values on the choreography context
  b subscribe for data notification (`publish()` and `subscribeFor()` arcs)
  c receive notifications from the CtxMg. In case of notifications, the interface notifies the KernelCI about them (`notify()`)
- the *CIProxyIn* interface mediates the interaction with the wrapped service and offers the KernelCI the API to invoke its operations (`callOp()`). Moreover, it notifies the KernelCI about the messages generated by the web service during its execution (`send()`).

Figure 10 Software architecture



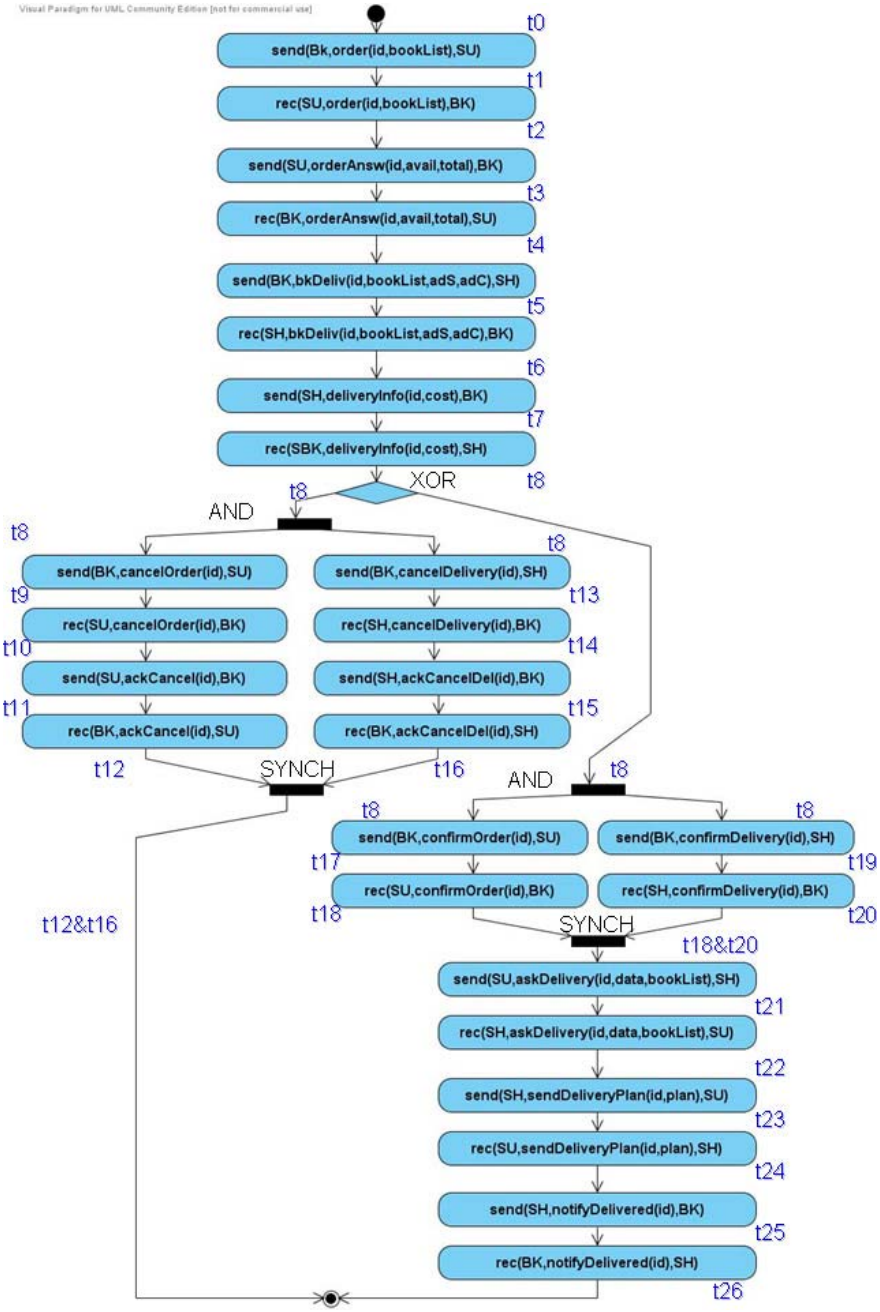
We designed the CIProxyOut and the CIProxyIn as interfaces in order to decouple the KernelCI from the technologies and the communication protocols used by the CtxMg and by the wrapped service. While the former abstraction enables the replacement of the CtxMg, the independence of the wrapped service enables the integration of web services, as well as of Java-based applications and other kinds of software components. Specifically, different implementations of the interfaces can be developed depending on the particular environment and technology used, without changing the business logic of the KernelCI. In our prototype, the CIProxyOut implementation (*CIProxyOutGiga*) exploits the APIs offered by GigaSpaces to subscribe for, publish and receive the relevant context information.

A web service can be coupled to a CI in different ways:

- if the web service is explicitly configured to participate in a composite service, web service handlers can be exploited [similar to Biornstad et al. (2006)] to efficiently manage the interaction between the wrapped service and the CI

- if a service-dependent configuration has to be avoided (e.g., in the case of a widely used public web service), the CI has to be developed as a web service itself. This means that the CI plays the role of a regular client of the web service to be wrapped and the composite service invokes the CI as its service supplier.

Figure 11 BookShopping choreography decorated with synchronisation tokens (see online version for colours)



8 Use case

As a use case for the exploitation of our framework, we consider an e-commerce service specified by the *BookShopping* choreography depicted in Figure 11. This choreography specification describes the messages to be exchanged by three web services, playing the roles of a *BookStore* (*BK*), a *supplier* (*SU*) and a *shipper* (*SH*), in order to provide a complex BookShopping service.

In the following, we describe the interaction flow specified by the choreography. For readability, we omit the parameters of the messages, which can be found in the figure: the BookStore orders a `bookList` by interacting with the supplier (`order()`, `orderAnsw()`) and asks the Shipper about the delivery cost, given the addresses of the supplier and of the customer (`bkDeliver()`, `deliveryInfo()`). In turn, the BookStore can confirm or delete the order by interacting with its business partners.

- In case of deletion, the BookStore starts a parallel messages exchange (`AND`) where it asks its partners for the deletion of the order (`cancelOrder()`, `cancelDelivery()`) and waits for an acknowledgment message from each of them (`ackCancel()`, `ackCancelDel()`); then the process terminates.
- In case of confirmation, a parallel messages exchange is started to confirm the order (`confirmOrder()`, `confirmDelivery()`); in this case, the Bookstore does not expect any acknowledgments from its partners.

When the supplier and the shipper have received the confirmation (after the SYNCH join), the supplier asks the shipper for the delivery of the goods, suggesting a date for the appointment on its site (`askDelivery()`). The shipper responds with a plan for the delivery (`sendDeliveryPlan()`) and after having delivered the items to the customer, it acknowledges the BookStore (`notifyDelivered()`) and the collaboration ends.

We suppose that the *NewSupplier* web service (henceforth, *NSU*) is interested to take part in this choreography, by playing the supplier role. Figure 12 shows the interaction protocol of NSU. It can be noticed that the protocol is a bit different from what is expected within the BookShopping choreography. In particular, the following mismatches hamper the integration of NewSupplier in the composite service:

- The signatures of the operations differ in operation name, order and name of arguments.
- In case of order deletion (`unreserve()`), NSU does not provide the acknowledgment message occurring in the choreography. This is an *extra message mismatch*.
- In case of order confirmation, NSU generates an acknowledgment message which is not expected in the choreography; *missing message* mismatch.
- Instead of the `askDelivery()` message prescribed by the choreography, NSU specifies the delivery request in two different messages: a `requestDelivery()` message and a notification of the date suggested for the appointment (parameter `when` in `reqTimeDelivery()`). This is a *message split* mismatch.

- Finally, NSU waits for a `notifyDelivered()` message in order to know whether the items have been delivered to the customer. Notice that, in the choreography, a similar message (`notifyDelivered()`) is exchanged between the BookStore and the Shipper, but it does not involve the Supplier. This is a *wrong sender* mismatch.

The above listed mismatches can be solved using our mediation framework without changing the interaction protocol of NSU. Specifically, Table 5 shows the data mapping table and Table 6 shows the message mapping table obtained by mapping data items and messages. Moreover, Table 7 shows the token mapping table.

Table 5 Data mapping table of web service NewSupplier (NSU) within the BookShopping composite service

#	*NSU business data*	*BookShopping business data*
1	ordId	id
2	ordList	bookList
3	answer	avail
4	price	total
5	when	data
6	deliveryPlan	plan

Table 6 Message mapping table of web service NSU within the BookShopping composite service

#	*NSU messages*	*BookShopping messages*
1	rec(book(ordList, ordId), BK)	rec(SU, order(id, bookList), BK)
2	send(bookResp(answer, price, ordId), BK)	send(SU, orderAnsw(id, avail, total), BK)
3	rec(unreserve(ordId), BK)	rec(SU, cancelOrder(id), BK)
4	-	send(SU,ackCancel(id), BK)
5	rec(confirmOrder(ordId), BK)	rec(SU, confirmOrder(id), BK)
6	send(ackConfirm(ordId), BK)	-
7	send(requestDelivery(ordList, ordId), SH) send(reqTimeDelivery(when, ordId),SH)	send(SU, askDelivery(id, data, bookList), SH)
8	rec(notifyPlan(deliveryPlan, ordId), BK)	rec(SU, sendDeliveryPlan(id, plan), SH)
9	rec(notifyDelivered(ordId), SH)	rec(BK, notifyDelivered(id), SH)

Table 7 Token mapping table of NSU within the BookShopping choreography

#	*WSA tokens*	*Target tokens*
1	so	t1
2	s1	t2
3	s2	t3
4	s3	t10
5	s4	t18
6	s7	t21
7	s8	t24

Figure 12 shows the interaction protocol of NSU decorated with the synchronisation tokens: the local tokens to be replaced with the target ones are showed in square brackets; e.g., in `[s4]t18`, `s4` is the local token and `t18` is the corresponding target one. Finally, Figure 13 shows a fragment of the `CI KB` of the NSU web service, after the application of the mapping rules described in Section 6. This fragment shows how the messages `send(requestDelivery(), SH)` and `send(reqTimeDelivery(), SH)` of NSU are mapped to the `send(SU, askDelivery(), SH)` choreography message.

Figure 12 NewSupplier (NSU) interaction protocol, decorated with local and global synchronisation tokens (see online version for colours)

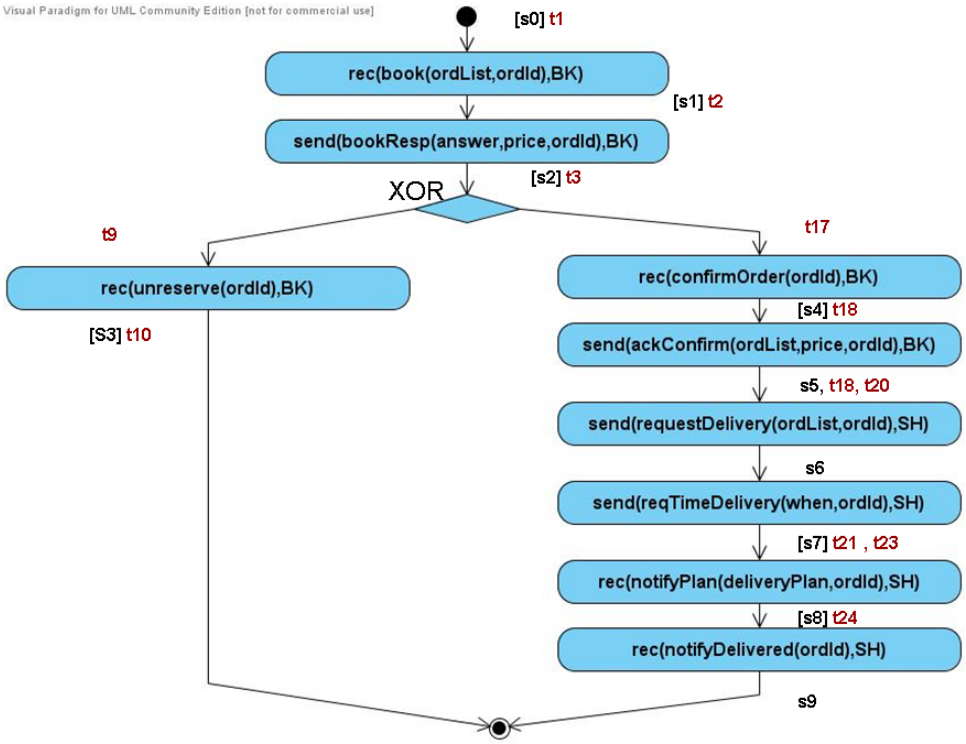


Figure 13 Fragment of the CI KB of the NSU web service

```
sendAction_requestDelivery
  prec:  s5 and t18 and  t20
  in: --
  out: s6
  op: requestDelivery(ordList, ordId)
  mappings: (id, ordId), (bookList, ordList)


sendAction_reqTimeDelivery
  prec:  s6
  in:  --
  out: data, t21
  op: reqTimeDelivery(when, ordId)
  mappings: (ordID, ordId), (when, data)
```

9 Evaluation of our framework

Our work aims at supporting the integration of heterogeneous software and the management of long-lived, asynchronous services, with minor performance requirements, because they automatise lengthy activities which often involve the human user. Thus, different from other work, we are not going to present numerical performance measures. In contrast, the most important features to be considered are the capability of handling legacy and heterogeneous software, and the scalability.

9.1 Heterogeneous software integration

Our framework addresses in a general way the problems of coordinating and mediating heterogeneous applications. For instance, we have applied it in an environment supporting the synchronisation of WSDL/SOAP and REST web services, as well as web applications (see Ardissono et al., 2009). This flexibility is supported by the following features:

- On the one hand, the CI architecture provides a high level of decoupling between the components of the framework. In particular, as described in Section 7, the CIProxyIN portion of the CI has to be developed for the individual software to be wrapped, by taking into account the offered APIs and its execution model. Once developed, that portion hides the peculiarities of the wrapped component.
- On the other hand, the event-based model implemented by the CtxMg offers the possibility to add new components and services (or to modify existing ones) in a seamless way. Even though in this article we assumed the compatibility of data types among the services, this assumption can be relaxed. For instance, nothing prevents the insertion of additional services to the composite service, with the purpose of performing data conversion among different formats. These converter services would work only on the choreography context and they would be transparent to the other services.

9.2 Scalability

The core of our framework borrows the scalability and performance properties from the middleware used to implement the CtxMg. In particular, GigaSpaces provides a scalable environment where clients can publish and read information. Moreover, it supports space replication and a data persistence mechanism based on Hibernate or on other widely used persistence tools. Thus, it is suitable for an efficient and scalable management of composite services. GigaSpaces offer immediate advantages because it supports messaging at the level of each participant of the composite service, instead of providing messaging services externally. Moreover, it supports the virtualisation of the shared space; therefore, it can handle scaling events implicitly, i.e., route requests to a different server if one is busy, without any intervention from the application. Notice however that our framework is not bound to GigaSpaces, and another middleware could be employed for managing the shared space. In fact, CIProxyOUT interface of the CI, which handles the interaction with the CtxMg, allows a seamless integration of a different middleware, as soon as it supports the interaction via publish and subscribe.

As each CI is devoted to the management of a specific web service, its impact on the integrated environment is limited and mainly depends on the number of events it publishes in the choreography context. Indeed, the required level of integration among web services determines the amount of context information to be propagated, and the workload on the CtxMg. However, this strongly depends on the kind of integrated environment to be managed.

In the following, we briefly compare the network traffic generated by our framework to that of a standard composite service.

- On the one hand, a CI has to retrieve one synchronisation token, and to produce one, for each inbound/outbound message of the wrapped web service; however, the token transmission is negligible if compared to the management of SOAP messages invoking WSDL operations.
- On the other hand, our framework propagates the business data as events, and the management of SOAP messages is limited to the interaction between each CI and its wrapped web service. Therefore, if the CI is installed on the same server hosting the web service, the impact on the network traffic is negligible. Moreover, the CI can be implemented, similar to Biornstad et al. (2006), as a web service handler, in which case the communication overhead is further reduced.

Anyway, the number of messages, compared to the point-to-point propagation of the business data via SOAP/WSDL messages, is limited because, according to the publish and subscribe pattern, each web service sends a message only once, regardless of the number of recipients. Notice also that, for each message, only the new and the revised business data is propagated in the choreography context. Differently, in standard web service composition, all the arguments of a WSDL operation are transmitted, including redundant information, such as, e.g., the BPEL correlation sets.

For instance, let's consider the network traffic generated by the BookShopping (*BK*), supplier (*SU*) and shipper (*SH*) web services, in the *BookShopping* choreography described in Chapter 8. As discussed in the following, the network traffic generated by our event-based model in this use case does not exceed the traffic produced in the standard choreography management.

Table 8 compares the number of data items exchanged by the web services if they interact in a traditional web service composition model via WSDL messages (*message-based* row), with the one obtained my managing them in our approach (*event-based* rows). Specifically:

- Row *message-based* reports the number of SOAP messages sent (*out*) and received (*in*) by each web service. The last column shows the sum of the previous figures, divided by two in order to take into account the fact that each WSDL message is counted twice (as an outbound message of the sender and as an inbound message of the receiver).
- Row *event-based – data* shows the number of data items published by the web service (*out*), or received as notifications (*in*), and their total number. As each event published or notified involves a piece of information that transits from the service (its CI) and the CtxMg, in this case the events are not duplicated.
- Row *event-based – tokens* shows the corresponding numbers of synchronisation tokens.

Table 8 Number of exchanged pieces of information

Interaction model	BK_{in}	BK_{out}	SU_{in}	SU_{out}	SH_{in}	SH_{out}	*TOT#*
Message-based	5	6	4	3	4	4	13
Event-based – data	3	4	3	3	5	2	20
Event-based – tokens	8	11	8	7	8	8	50

As previously discussed, messages and events have a rather different size, which makes the former heavier to handle than the latter. In order to approximate their average impact on network traffic, we propose to weight them as follows:

- A synchronisation token is an atomic information item; therefore, it is assigned a weight = 1.
- A data item can be more or less complex. As a rought estimate, we set its weight = 4.
- A typical SOAP message is composed of an operation name (weight=1), two arguments (weight = 8) and the SOAP infrastructure (weight = 2), for a total weight = 11.

Table 9 reformulates the findings described in Table 8 for the choreographed service, by taking such weights into account. As shown, the estimated global traffic weight of the message-based approach (143) is a bit higher than the one estimated for the event-based approach (80 + 50 = 130).

Table 9 Weight of exchanged pieces of information

Interaction model	*TOT#*	*Global weight*
Message-based	13	143
Event-based – data	20	80
Event-based – tokens	50	50

10 Related work

Our work extends the previous research on the management of bi-lateral web service interaction (e.g., see Yellin and Strom, 1997; W3C, 2002; Benatallah et al., 2005) by supporting the mediation of crossprotocol mismatches between sets of web services. For instance, our model abstracts from the identities of the services which generate the business data of a composite application. Thus, it can handle protocol mismatches concerning the data flow between various cooperating web services. Moreover, with respect to the work in Biornstad et al. (2006) our framework supports not only the runtime enforcement of an interaction protocol, but also the conciliation of mismatches at the side of the participant web service.

As far as the web service coordination is concerned, we build on the previous research on event condition action execution models (Kappel et al., 1998; Milosevic et al., 2006; Ma et al., 2007) in order to support the context-dependent web service execution. However, thanks to the introduction of the choreography context, our work supports the coordination of both orchestrated services (corresponding to the workflows handled in ECA models) and choreographed ones. This generality provides a natural

support to the decentralisation of services, which has been studied in several projects in order to address the overhead of a centralised coordination based on complex orchestration logic (e.g., see Chafle et al., 2004; Zirpins et al., 2004).

Our event-based coordination model is similar to the approach adopted in Monsieur et al. (2007) and Snoeck et al. (2004). However, we relax the condition that enables the interaction of the web services through the use of the CI. As long as the basic synchronisation constraints are observed, the CI overcomes the problem of sending and receiving data as exactly specified in the WSDL interface.

The work about choreography specification languages focuses on the design of composite services and on the derivation of the participants' local interaction protocols. For instance, see Let's dance (Zaha et al., 2006a and 2006b), Web Service Choreography Interface (WSCI) (Arkin et al., 2002), Web Services Choreography Description Language (WS-CDL) (W3C, 2005), Chor (Zongyan et al., 2007) and business process modelling languages, such as Business Process Modelling Notation (OMG Object Management Group, 2008). Our goal is different, i.e., supporting the set-up and execution of a composite service.

The semantic web research proposed various mediation frameworks to manage choreographed services and solve data and interaction protocol mismatches. For example, see Mrissa et al. (2007), and Bussler (2006), the Web Service Modelling Ontology (WSMO) (DERI International, 2005), the Web Service Execution Environment (WSMX) (Zaremba et al., 2005; DERI International, 2008), or the SWWS project (Williams et al., 2006). While our work does not focus on data mismatches, it addresses protocol mismatches in a more flexible way: e.g., with respect to SWWS, we propose a lightweight abstraction approach, based on the propagation of tokens. Moreover, our framework supports the decentralised reconciliation of interaction protocol mismatches. In contrast, WSMX and Bussler (2006) propose a centralised solution, based on an external choreography management service (e.g., see Cimpian and Mocan, 2006).

Different from Ponnekanti and Fox (2004), Benatallah et al. (2005), and Motahari-Nezhad et al. (2007), we do not attempt to recognise signature and protocol mismatches automatically; however, we provide a semi-automatic support to the generation of adapters. In particular, our framework deals explicitly with the protocol mismatches described in Benatallah et al. (2005), leaving data heterogeneity issues apart. Moreover, our mediation model is more general than Motahari-Nezhad et al. (2007), as we address the mediation among several services taking part in a multi-party interaction.

The issue of synthesising adapters for incompatible protocols has been deeply studied in the area of software composition and component-based software engineering (e.g., see Brogi and Popescu, 2006; Xiong and Weishi, 2005; Bracciali et al., 2003; Hemer, 2005; Jiao and Mei, 2006). In that area, Mouakher et al. (2006) proposes a generation of adapters that realise the matching between components specified by their required and provided interfaces. The components are described using high-level UML 2.0 interfaces automatically transformed to formal B specifications (Abrial, 1996). Moreover, in Autili et al. (2008), after the derivation of a central adapter, individual local adapters for each component are derived in order to satisfy the required behaviour. With respect to such works, our framework does not support a fully automated generation of adapters. However, thanks to the context-based synchronisation of services, it deals with a larger number of protocol mismatches; e.g., consider the *wrong sender mismatch* described in Section 2, which involves the identity of the senders of messages.

Li et al. (2008) presents a semi-automatic approach to the conciliation of interaction protocol mismatches, similar to the one adopted in our framework, but dealing with fewer types of protocol mismatch. Starting from a classification of mismatch patterns in the context of heterogeneous services composition (Li et al., 2007), the authors propose a set of mediation patterns supporting the conciliation of simple mismatches and a set of heuristic rules for the selection of the patterns to be combined in the conciliation of more complex mismatches.

A different perspective is taken in Kongdenfha et al. (2009): starting from the patterns identified in Benatallah et al. (2005), that work presents an approach to protocol mediation based on aspect-oriented programming (AOP). The basic consideration is that adaptation logic is a cross-cutting concern, which means that from the developer point of view it is transversal to the other functional concerns of the service. The approach is based on the definition of aspect templates that, when instantiated, generate a collection of adaptation aspects that will be woven into the service at runtime. Also in this case, we can say that our work enables the solution of a larger set of mismatches.

BPEL4Chor (Decker et al., 2007) separates the specification of control flow dependencies, interaction links and concrete configurations of data formats and port types, in order to support the adaptation of interaction protocols within a choreographed service. However, the control flow within a choreography is defined from the viewpoint of the participant web services, which have to explicitly adapt to each other. Our approach is more flexible and synthetic, as it enables the web services to adapt to the composite service, regardless of the web services filling the other roles of the choreography.

11 Conclusions

This article has presented a mediation framework supporting the integration of web services in composite services and the conciliation of interaction protocol mismatches. The framework is based on the introduction of:

a a Context Management Service which handles the context of the composite service and propagates the relevant business data and synchronisation information to the participant web services

b a set of knowledge-based adapters which decouple the interaction between web services and manage them as event-based systems synchronising in a shared context.

By decoupling the web service interaction, our model solves several interaction protocol mismatches, ranging from differences in the signatures of the messages, to the order and number of the messages to be exchanged, including cross-protocol mismatches involving more than two interacting peers.

References

Abrial, J.R. (1996) *The B-Book*, Cambridge University Press.

Ahuja, S., Carriero, N. and Gelernter, D. (1986) 'Linda and friends', *IEEE Computer*, Vol. 19, No. 8, pp.26–34.

Ardissono, L., Goy, A., Petrone, G. and Segnan, M. (2009) 'SynCFr: synchronization collaboration framework', in *Proc. of 5th Conference on Internet and Web Applications and Services (ICIW 2009)*, pp.18–23, IEEE, Venezia, Italy.

Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I. and Zimek, S. (2002) 'Web service choreography interface 1.0', available at http://ifr.sap.com/wsci/specification/wscispecp10.html.

Autili, M., Mostarda, L., Navarra, A. and Tivoli, M. (2008) 'Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems', *Journal of Systems and Software (JSS)*.

Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R. and Toumani, F. (2005) 'Developing adapters for web services integration', in *Proc. 17th Conf. on Advanced Information Systems Engineering (CAiSE'05)*, Porto, Portugal.

Biornstad, B., Pautasso, C. and Alonso, G. (2006) 'Enforcing web services business protocols at run-time: a process-driven approach', *Int. Journal of Web Engineering and Technology*, Vol. 2, No. 4, pp.396–411.

Bracciali, A., Brogi, A. and Canal, C. (2003) 'A formal approach to component adaptation', *The Journal of Systems and Software*, Vol. 74, No. 1, pp.45–54.

Brogi, A. and Popescu, R. (2006) 'Automated generation of BPEL adapters', in *LNCS n. 4294: Service-Oriented Computing (ICSOC 2006)*, pp.27–39, Berlin Heidelberg.

Bussler, C. (2006) 'Message mediation in composite web services', *Int. Journal of Web Engineering and Technology*, Vol. 2, No. 4, pp.373–395.

Chafle, G., Chandra, S., Mann, V. and Nanda, M.G. (2004) 'Decentralized orchestration of composite web services', in *Proc. of 13th Int. World Wide Web Conference (WWW'2004)*, pp.134–143, New York.

Cimpian, E. and Mocan, A. (2006) 'WSMX process mediation based on choreographies', in *LNCS n. 3812: Business Process Management Workshops*, pp.130–143, Berlin Heidelberg.

Decker, G., Kopp, O., Leymann, F. and Weske, M. (2007) 'BPEL4Chor: extending BPEL for modeling choreographies', in *IEEE Int. Conf. on Web Services (ICWS'07)*, pp.296–303, IEEE Computer Society, Salt Lake City, Utah.

DERI International (2005) 'Web service modeling ontology', available at http://www.w3.org/Submission/WSMO/.

DERI International (2008) 'Web service modelling execution environment', available at http://www.wsmx.org/.

GigaSpaces (2008) 'GigaSpaces SBA', available at http://www.gigaspaces.com/proverview.html.

Hemer, D. (2005) 'A formal approach to component adaptation and composition', in *ACSC '05: Proceedings of the Twenty-Eighth Australasian Conference on Computer Science*, pp.259–266, Australian Computer Society, Inc., Darlinghurst, Australia, Australia

Jiao, W. and Mei, H. (2006) 'Automating integration of heterogeneous cots components', in Morisio, M. (Ed.): *ICSR, Volume 4039 of Lecture Notes in Computer Science*, pp.29–42, Springer.

JINI (2008) 'JavaSpaces specification', available at http://www.jini.org/wiki/JavaSpacesSpecification.

Kappel, G., Rausch-Schott, S. and Retschitzegger, W. (1998) 'Coordination technology for collaborative applications – organizations, processes, and agents', in Conen, W. and Neumann, G. (Eds.): *LNCS n. 1364: 1996 ASIAN 1996 Workshop*, pp.359–362, London.

Kongdenfha, W., Motahari-Nezhad, H.R., Benatallah, B., Casati, F. and Saint-Paul, R. (2009) 'Mismatch patterns and adaptation aspects: a foundation for rapid development of web service adapters', *IEEE Transactions on Services Computing*, Vol. 2, No. 2, pp.94–107.

Li, X., Fan, Y. and Jiang, F. (2007) 'A classification of service composition mismatches to support service mediation', in *GCC*, pp.315–321.

Li, X., Fan, Y., Wang, J., Wang, L. and Jiang, F. (2008) 'A pattern-based approach to development of service mediators for protocol mediation', in *WICSA*, pp.137–146.

Lu, R. and Sadiq, S. (2007) 'A survey of comparative business process modeling approaches', in *Proc. of 10th Int. Conf. on Business Information Systems (BIS), LNCS 4439*, pp.82–94, Poznan, Poland.

Ma, D.C., Lin, J.Y.C. and Orlowska, M. (2007) 'Automatic merging of work items in business process management systems', in *Proc. of 10th Int. Conf. on Business Information Systems (BIS'07)*, pp.14–28, Poznan, Poland.

Milosevic, Z., Orlowska, M. and Sadiq, S. (2006) 'Linking contracts, processes and services: an event-driven approach', in *Proc. of the IEEE International Conference on Services Computing*, pp.390–397, Washington, DC.

Monsieur, G., Snoeck, M. and Lemahieu, W. (2007) 'Coordinated web services orchestration', in *IEEE Int. Conf. on Web Services (ICWS'07)*, pp.775–783, IEEE Computer Society, Salt Lake City, Utah.

Motahari-Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F. and Casati, F. (2007) 'Semiautomated adaptation of service interactions', in *Proc. of 16th Int. World Wide Web Conference (WWW'2007)*, pp.993–1002, Banff, CA.

Mouakher, I., Lanoix, A. and Souquires, J. (2006) 'Component adaptation: specification and verification', in *WCOP 2006: Proceedings of the Eleventh International Workshop on Component-Oriented Programming*.

Mrissa, M., Ghedira, C., Benslimane, D., Maamar, Z., Rosenberg, F. and Dustdar, S. (2007) 'A context-based mediation approach to compose semantic web services', *ACM Transaction on Internet Tecnology (TOIT), Special Issue on Semantic Web Services*, Vol. 8, No. 1, Article 4.

OASIS (2005) 'OASIS web services business process execution language', available at http://www.oasis-open.org/committees/documents.php?wg abbrev=wsbpel.

Oasis (2009) 'Web services coordination (WS-coordination)', available at http://docs.oasisopen.org/ws-tx/wscoor/2006/06.

OMG Object Management Group (2008) 'Business process modeling notation (BPMN) information', available at http://www.bpmn.org.

Ortiz, G., Hernandez, J. and Clemente, P.J. (2006) 'Preparing and re-using web services for choreography', *Int. Journal of Web Engineering and Technology*, Vol. 2, No. 4, pp.307–334.

Papadopoulos, G.A. and Arbab, F. (1998) 'Coordination models and languages', in *Advances in Computers*, pp.329–400, Academic Press.

Papazoglou, M.P. and Georgakopoulos, D. (Eds.) (2003) 'Service-oriented computing', *Communications of the ACM*, Vol. 46.

Papazoglou, M.P. and Van Den Heuvel, W-J. (2006) 'Service-oriented design and development methodology', *Int. Journal of Web Engineering and Technology*, Vol. 2, No. 4, pp.412–442.

Peltz, C. (2003) 'Web services orchestration and choreography', *Innovative Technology for Computing Professionals*, Vol. 36, No. 10, pp.46–52.

Ponnekanti, S.R. and Fox, A. (2004) 'Interoperability among independently evolving web services', in *LNCS n. 3231: Middleware 2004*, pp.73–85, Berlin Heidelberg.

Snoeck, M., Lemahieu, W., Goethals, F., Dedene, G. and Vandenbulcke, J. (2004) 'Events as atomic contracts for component integration', *Data Knowledge & Knowledge Eengineering*, Vol. 51, pp.81–107.

van der Aalst, W., Ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A. (2003) 'Workflow patterns', *Distributed and Parallel Databases*, Vol. 14, No. 1, pp.5–51.

van der Aalst, W., Ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A. (2008) 'Conformance checking of service behavior', *ACM Transactions on Internet Technology (TOIT), Special Issue on Service-Oriented Computing*, Vol. 8, No. 3, Article 13.

W3C (2001) 'Web services definition language', available at http://www.w3.org/TR/wsdl.

W3C (2002) 'Web services conversation language (WSCL)', available at http://www.w3.org/TR/wscl10.

W3C (2005) 'Web services choreography description language version 1.0', available at http://www.w3.org/TR/ws-cdl-10/.

Williams, S.K., Battle, S.A. and Cuadrado, J.E. (2006) 'Protocol mediation for adaptation in semantic web services', in *LNCS n. 4011: The Semantic Web: Research and Applications*, pp.635–6495, Berlin Heidelberg.

Xiong, X. and Weishi, Z. (Eds.) (2005) 'The current state of software component adaptation', in *IEEE Computer Society, First International Conference on Semantics, Knowledge and Grid (SKG'05)*.

Yellin, D.M. and Strom, R.E. (1997) 'Protocol specifications and component adaptors', *ACM Transactions on Programming Language and Systems*, Vol. 19, No. 2, pp.292–333.

Zaha, J.M., Barros, A.P., Dumas, M. and ter Hofstede, A.H.M. (2006a) 'Let's dance: a language for service behavior modeling', in *OTM Conferences 2006 – Cooperative Information Systems (CoopIS) 2006 Int. Conf*., pp.145–162, Montpellier, France.

Zaha, J.M., Dumas, M., ter Hofstede, A.H.M., Barros, A.P. and Decker, G. (2006b) 'Service interaction modeling: bridging global and local views', in *Proc. of 10th IEEE International EDOC Conference (EDOC 2006), The Enterprise Computing Conference*, pp.45–55, Hong Kong, China.

Zaremba, M., Moran, M., Haselwanter, T., Lee, H-K. and Han, S-K. (2005) 'D13.4v0.3 WSMX architecture', available at http://www.wsmo.org/TR/d13/d13.4/v0.3/.

Zirpins, C., Lamersdorf, W. and Baier, T. (2004) 'Flexible coordination of service interaction patterns', in *Proc. of 13th Int. World Wide Web Conference (WWW'2004)*, pp.49–56, New York.

Zongyan, Q., Ziangpeng, Z., Chao, C. and Hongli, Y. (2007) 'Towards the theoretical foundation of choreography', in *Proc. of 16th Int. World Wide Web Conference (WWW'2007)*, pp.973–980, Banff, CA.

Notes

1 The situation is similar for orchestrated web services, if the web service suppliers to be invoked are willing to adapt to the orchestrator. This situation might hold in enterprise integration scenarios, where service consumers establish long-term relationships with service providers.

2 The token-based representation relaxes the synchronisation constraints of the XOR case. However, this is not a problem because, if a web service behaves coherently with its own interaction protocol, it generates only one of the alternative messages. The analysis of the conformance between business logic and interaction protocol is out of the scope of our work. See, e.g., van der Aalst et al. (2008).

3 The local tokens are included in the precondition as follows:

*If *LM* is located immediately after an unmapped message LM_i, then the precondition must be ANDed with the output token of LM_i.

*If *LM* is located on an arc which joins a set of parallel flows (SYNCH), originating from the LM_1, ..., LM_n messages, the precondition must be ANDed with the conjunction of the output tokens of LM_1, ..., LM_n.

*If *LM* is located on an arc which joins a set of alternative flows (MERGE), originating from the LM_1, ..., LM_n messages, the precondition must be ANDed with the disjunction of the output tokens of LM_1, ..., LM_n.