

# An Event-based Model for the Management of Choreographed Services\*

Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan

Dipartimento di Informatica - Università di Torino  
Corso Svizzera 185, 10149 Torino - Italy  
{liliana,furnari,goy,giovanna,marino}@di.unito.it  
<http://www.di.unito.it>

**Abstract.** We propose a mediation framework supporting a flexible management of choreographed services. The framework separates the management of the business logic of a service from the message flow details reported in the choreography specification. This abstraction step is achieved by introducing an event-driven management of Web Services and by exploiting a rich coordination context which includes business data and synchronization information. By separating activity management issues from communication details, our framework facilitates the management of interaction protocol mismatches between Web Services.

Copyright by Springer Verlag. LNCS 5183, E-commerce and Web technologies, 9th International Conference, EC-Web 2008. G. Psaila and R. Wagner(Eds.).

## 1 Introduction

In Web Service composition, choreographies are introduced to specify applications whose business logic is managed in a decentralized way; see [1]. A choreographed service results from the cooperation of multiple Web Services (WSs) which coordinate the activities to be performed via message passing.

A choreography specification represents a contract to be accepted by the WSs which want to cooperate to the management of a composite service. The contract enables the WSs to agree on the activities to be performed, preventing run time errors and violations of the business logic of the composite service. However, in order to participate as a role filler in the service, a WS has to adapt its own interaction protocol to the choreography specifications, by solving the occurring mismatches. For instance, there might be differences in operation names, parameter lists and message flow. This means that a WS participating to more than one composite service might be provided with different WSDL [2]

---

\* This work was partially funded by projects WS-Diamond (IST-516933) and QuaD-RAnTIS (MUR).

interfaces. Moreover, for each choreography, an adapter should be developed in order to enforce the specified order in the generation and reception of messages.

In order to address such issues, we propose an event-driven choreography management framework supporting the mediation of interaction protocols within a composite service. Our model is based on the idea that, during the execution of a composite service, it is possible to abstract from several message flow details specified in the choreography, provided that the data dependencies and synchronization constraints imposed by the business logic of the service are respected. In order to support the cooperation between Web Services and their synchronization, our model introduces a Choreography Coordination WS which manages the Choreography Coordination Context by collecting business data and synchronization information and by propagating them to the interested Web Services according to the Publish and Subscribe pattern. Our framework also enforces an event-driven execution of activities [3] within WSs, in order to enable them to autonomously operate, as soon as they are provided with the data items they need, and the related synchronization constraints are satisfied.

In the following, Section 2 presents the motivations of our work and Section 3 describes our choreography management model. Section 4 describes the event-based representation of a choreography that is the basis of our proposal. At last, Section 5 positions our work in the related work and concludes the paper.

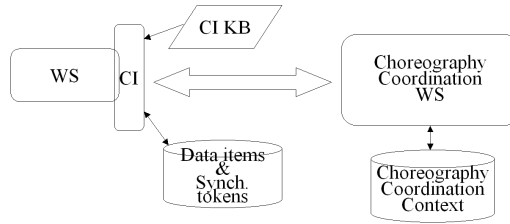
## 2 Motivations of Our Work

A choreography specification describes the business logic of a composite service in a message-oriented way, by specifying the expected interaction flow between the cooperating WSs. As messages invoke WSDL operations, this specification enforces the execution of activities, therefore supporting the WS coordination. However, the WSs participating to an application might offer operations which differ from those specified in the messages of the reference choreography. Thus, in order to fill a role within a composite application, the interaction protocol of a WS has to be adapted to the message flow specified in the choreography.

We assume that the choreography and the WS interaction protocols are described as UML activity diagrams. This notation is suitable for describing the interaction flow. In fact, UML activity diagrams can be mapped to Petri Nets, which are a reference model for the specification of processes; see [4].

Even though a Web Service complies with the representation and meaning of the business data handled in the application, various protocol mismatches are possible. For instance, the names and parameter list of the WSDL operations might differ from those occurring in the choreography specification. Moreover, the expected number and order of messages might differ from the interaction protocol of the WS. Although ad-hoc adapters can be developed to solve protocol mismatches, a general solution to the individual Web Services interaction protocol adaptation should be identified.

Indeed, the information to be specified in order to support the WS coordination within a choreographed service consists of: the business data to be handled,



**Fig. 1.** Architecture of our Choreography Coordination Model.

the operations to be performed on data items, the data dependencies between operations and possibly other pre-conditions on operations. However, a choreography specification embeds this information in a detailed message flow specification which conveys additional constraints, such as the signatures of the operations. We claim that a general run-time model for choreography management can be developed by abstracting from the flow details imposed by message-oriented coordination and by focusing on data-dependencies and synchronization. This approach deeply changes the role of the choreography specification, which does not represent any more the run-time behavior of the cooperating WSs. In fact, the choreography is used when the composite service is set up, as a reference to build the run-time model of the service and to help solving mismatches between the protocols of the individual WSs filling a particular role within the choreography.

### 3 Choreography Management Model

#### 3.1 Architecture

We propose to manage the cooperation among WSs by handling a Choreography Coordination Context which supports the coordination of the WS activities within the composite application. Figure 1 depicts the architecture of our choreography management model:

- The Choreography Coordination WS (ChCoordWS) manages the Choreography Coordination Context of the application, which maintains:
  - Basic coordination information; e.g., the identifier of the choreography instance which the cooperating Web Services participate to; see [5].
  - The data items to be shared between WSs (business data).
  - The synchronization constraints that have to be satisfied during the execution of the application. As described in Section 4, the choreography specification and the interaction protocol of the WS are translated to an event-based representation where synchronization constraints are represented as tokens. Thus, the satisfaction of a synchronization constraint is described by making the corresponding token available in the Choreography Coordination Context.

- In order to enable the cooperating WSs to perform activities autonomously, our model replaces the message-based invocation of WSDL operations with an event-driven action execution [3]. Each cooperating WS is thus wrapped by an adapter, the Communication Interface (CI), which activates the execution of the services within the WS, on the basis of the available context information and of the existing synchronization constraints. The CI blocks the interaction with the other WSs in order to avoid the direct invocation of WSDL operations; moreover, it manages the interaction between the WS and the ChCoordWS.

The ChCoordWS collects business data and synchronization information and propagates them to the CIs of the interested Web Services according to the Publish and Subscribe pattern, supporting distributed caching. Each CI receives from the ChCoordWS the information it had subscribed for, as soon as it becomes available. Moreover, when the wrapped WS generates some data items, or it satisfies a synchronization constraint, the CI publishes such information to the ChCoordWS, which notifies the interested WSs.

Notice that we have opted for a centralized management of the Context information because it supports the data consistency management (e.g., in the propagation of changes in the business data). In contrast, we have adopted a distributed management of the Web Service activities to minimize the amount of application dependent information which the ChCoordWS has to manage.

We are implementing our model by exploiting Web Services and message handlers for the development of the CIs. Moreover, we are implementing the ChCoordWS by exploiting the GigaSpaces [6] middleware, which supports the propagation of data items according to the Linda [7] tuple space model. GigaSpaces takes care of logging the publication and notification messages occurring during the execution of the application; as such, it provides a trace which can be utilized for error management purposes.

### 3.2 Action-based WS Representation

The event-driven activity execution is based on a representation of activities (WSDL operations) as the actions of an autonomous agent. The CI has a knowledge base (CI KB) storing the information needed to support the WS execution; see Figure 2. The CI KB stores the following types of information:

- The specification of the actions which the WS has to perform during its execution as the filler of a role of the application. In a composite application, a WSDL operation might be invoked more than once. Thus, given a WSDL operation, for each possible invocation, an `Actionx` slot is defined to describe the invocation setting. The slot specifies the following information:
  1. `in`: this field specifies the business data items (as described in the choreography specification) needed as input parameters for the execution of the WSDL operation.
  2. `out`: this field specifies the business data items and synchronization tokens produced by the operation (if any).

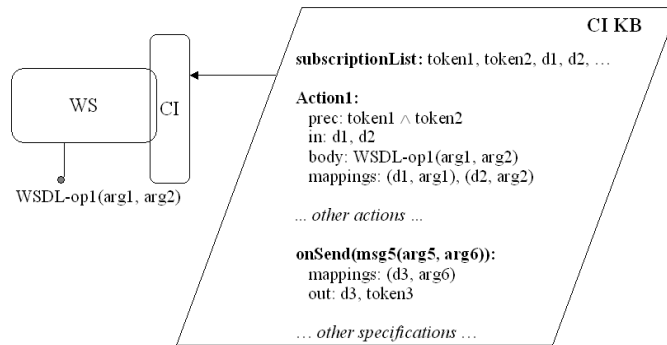


Fig. 2. Architecture of a WS supplier.

3. **mappings**: this field maps the business data to the input/output arguments of the operation.
4. **prec**: this field describes the precondition of the action, i.e., the synchronization constraints which must be satisfied in order to enable the execution of the operation. The preconditions are represented as boolean expressions on synchronization tokens.

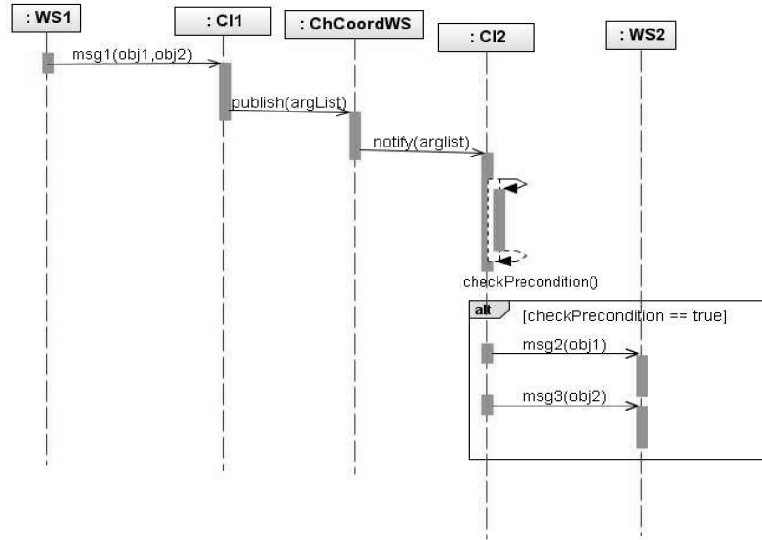
For instance, **Action1** in the figure describes preconditions and input parameters of the WSDL operation named **WSDL-op1**. The operation can be executed only after the synchronization constraints represented by *token1* and *token2* have been satisfied. Moreover, the arguments of the action (*arg1* and *arg2*) have to be bound to the *d1* and *d2* business data items.

- The CI KB also specifies the subscription list of the WS (**subscriptionList**). This is the list of business data and synchronization tokens relevant for the WS execution (i.e., occurring in the input parameters of actions and in their preconditions).
- Finally, for each outbound message *msg* belonging to the interaction protocol of the WS, an **onSend(msg)** slot specifies in its **out** field the business data items and the synchronization tokens to be published in the Choreography Coordination Context when the message is generated (plus the mappings between business data and message parameters). For example, the CI KB in Figure 2 specifies that, when the WS generates **msg5(arg5, arg6)**, the *d3* data item and the *token3* synchronization token must be published.

### 3.3 Event-driven WS Execution

The life cycle of a choreographed application includes an initialization and a management phases. At initialization time, the CI of each cooperating WS registers in the Choreography Coordination Context by sending a registration request to the ChCoordWS, according to the WS-Coordination specifications [5].

At runtime, the CI mediates the interaction between the WS, the ChCoordWS and the other cooperating WSs by receiving the messages from the ChCoordWS and by intercepting the outbound messages of the WS.



**Fig. 3.** Messages exchanged by WSs at choreography management time.

- When a WS, following its own interaction protocol, tries to invoke a WSDL operation on another WS, the CI absorbs the outbound message. Then, the CI extracts from the message the values of the output parameters and it generates the synchronization tokens, according to the `onSend` specifications in the CI KB. Finally, it publishes the values in the Choreography Coordination Context, by notifying the ChCoordWS. We have depicted this interaction in Figure 3:<sup>1</sup> WS1 tries to invoke operation `msg1` on the `obj1` and `obj2` actual parameters (`msg1(obj1, obj2)`); CI1 captures the message and publishes `obj1`, `obj2` and the related synchronization tokens, by sending ChCoordWS the `publish(arglist)` message.
- When the ChCoordWS receives some data items or tokens, it stores them in the Choreography Coordination Context. Then, it 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 synchronization tokens, it stores them locally and checks whether any `Action` of the Web Service can be performed. If any `Action` has all the input parameters instantiated, and its preconditions are satisfied, the CI invokes the associated WSDL operation on the WS and retrieves the result. In Figure 3, we have depicted this situation by showing that, when receiving `obj1` and `obj2`, CI2 invokes, respectively, the `msg2(obj1)` and `msg3(obj2)` operations on WS2.

<sup>1</sup> In this example, for simplicity, we assume that business data items and local parameters of the interaction protocols have the same names.

By mediating the interaction between WSs, our model solves several protocol mismatch problems; e.g.:

- Mismatches in the operation names, number of parameters, and number of operations to be invoked; e.g., a WS might offer a WSDL operation performing a complex operation, but the client might try to separately invoke more than one operation for that purpose (or viceversa).
- Unexpected or missing acknowledgment messages; e.g., a WS might wait for an acknowledgment before continuing the execution of activities, but the interaction protocol of the client might not include such a message; viceversa, the client might send a message which should be ignored by the WS provider.
- Mismatches concerning the expected senders of messages; e.g. a WS might expect to be invoked by a certain client in order to perform an operation, but in the composite application another WS might send the invocation message. In our model, the propagation of business data hides the identity of the WS which has produced it; therefore, the mismatches involving the source or destination of messages can be ignored, provided that the needed business data is produced by other WSs.

## 4 Setting up a Choreographed Application

In order to participate to a composite application, a WS must be wrapped by a CI and the CI KB must be configured according to the interaction protocol of the WS and the choreography specifications. By mapping the interaction protocol to the choreography messages, a correspondence is defined between the WS execution and the progress in the execution of the application. The mapping includes the following steps:

- First, the parameters of the messages occurring in the interaction protocol of the WS have to be mapped to the business data specified in the choreography. This task produces the *data mapping table*.
- Second, the inbound and outbound messages of the interaction protocol have to be mapped to the corresponding choreography messages (or sets of messages). If there is no correspondence, the messages must be positioned in a point of the choreography where they can be handled. This task produces a *message mapping table* which relates (possibly empty sets of) messages of the WS to (possibly empty sets of) choreography messages.

Given the mapping tables, the CI KB can be generated by defining an `Action` slot for each inbound message and an `onSend` slot for each outbound message of the WS interaction protocol. In order to facilitate the generation of such slots, we assume that both the choreography specification and the interaction protocol of the WS are translated to an internal format which makes synchronization information explicit: the *token-based representation*.

#### 4.1 Token-based Choreography/Protocol Representation

The UML activity diagrams describing a choreography specification and an interaction protocol can be translated to their token-based representations by adding an output synchronization token for each activity node of the diagrams, i.e., for each message.<sup>2</sup> The association of a token to a message means that, when the message is sent/received, the related synchronization constraint is satisfied.

However, this algorithm introduces redundant tokens because it does not take into account data dependencies between messages. Two types of dependencies can be considered in order to simplify the token-based representation: a) the dependencies among the messages producing data items and those having such parameters as input ones; b) the dependencies between messages which change the value of the data items. If this kind of information is specified (e.g., in WS-CDL [8]), the token-based representation can be simplified automatically; otherwise, the redundant tokens must be manually removed.

#### 4.2 Generation of the CI KB

Starting from the mapping tables and the token-based representations of the interaction protocol and of the choreography specification, the CI KB can be automatically generated. Specifically:

- For each expected invocation of a WSDL operation, an **Action** slot has to be defined:
  - The **in** and **out** fields are specified by retrieving input and output parameters from the WSDL operation and by replacing them with the corresponding business data, as reported in the *data mapping table*. The same information is utilized to specify the **mappings** field.
  - The preconditions have to include the synchronization tokens which must be available when the WSDL operation can be performed, according to the local interaction protocol and to the choreography specification. The idea is that of including in the preconditions the synchronization information imposed by the choreography and to ignore local tokens, unless these represent additional synchronization constraints to be satisfied. Thus, the preconditions must include the input tokens of the (set of) choreography messages to which the **Action** corresponds. Moreover, in some cases (e.g., where there is not a one-to-one mapping between WSDL operations and choreography messages), the preconditions must also include some local tokens aimed at synchronizing the operations within the interaction protocol of the WS.<sup>3</sup>

<sup>2</sup> Moreover, as far as the choreography is concerned, two output tokens must be added for each decision node where the actor making the decision differs from the actors sending the first messages after the decision.

<sup>3</sup> The presence of additional synchronization constraints can be associated to unsolvable protocol mismatches; e.g., mismatches causing deadlocks in the choreography. We assume that the WS administrator takes care of checking that the business logic of the WS does not violate any constraints of the choreographed application; therefore, the additional constraints that we consider here are not problematic.

- For each outbound message *msg* of the WS, an `onSend(msg)` slot has to be defined in the CI KB. The slot must specify the data items generated or modified by the WS; i.e., the output parameters of the message. Moreover, the slot must include the synchronization tokens to be generated by the CI, if any. The specification of the fields of the slot is similar to the one described for the `Action` slots.
- The subscription list is composed of the input parameters of the `Actions` and of the synchronization tokens appearing in their preconditions.

## 5 Conclusions and Related Work

We have presented a choreography coordination model which addresses the adaptation of Web Services (WSs) to the business logic of the application, and the management of interaction protocol mismatches. The model is based on an event-driven execution of WS suppliers and on the introduction of a Choreography Coordination WS managing a shared coordination context.

Being event-driven, our model bears some relation to a Complex Event Processing architecture, but we focus on the execution of a specific choreography where applications with complex protocols interact with each other, not on the activation of applications in response to the detection of complex events.

The model presented in this paper extends conversational models, initially defined to support one-to-one communication between Web Services (e.g., [9–11]), to the management of many-to-many conversations.

Although several choreography specification languages have been defined (e.g., WSCI [10] and WS-CDL [8]), to our knowledge, no choreography management model has been introduced which supports a flexible and lightweight execution of a choreographed application. Our proposal also tries to provide an answer to this need.

Some mediation frameworks have been designed in order to enable the interaction between WSs; e.g., WSMO [12], WSMX [13], or the proposal by Benatallah and colleagues [14]. However, such frameworks are mainly focused on data mediation and they propose complex solutions as far as interaction protocol mediation is concerned. For instance, WSMX [13] introduces an external choreography management service which, similar to the orchestrator of a composite application, monitors the evolution of the execution state of the application and instructs participants about how to continue the interaction.

In other projects, the overhead of centralized choreography coordinators is avoided, but the choreographed service is generated as an indirect product of the pre-compiled behavior of the cooperating Web Services, without any support. In such cases, the exchange of messages between the cooperating WSs only depends on their interaction protocols, which are adapted to the choreography specification in order to enable the correct execution of the application.

Our approach is similar to the BECO system described in [15], which uses an event-based coordination paradigm. However, we relax the condition that enables the interaction of the Web Services in a choreography through the use of the

Communication Interface component (CI). As long as the basic synchronization constraints are observed, in fact, the CI overcomes the problem of sending and receiving data as exactly specified in the WSDL interface.

[16] describes an interesting approach providing semi-automated support to identify and resolve mismatches between service interfaces and protocol, and for generating adapter specification. That approach offers service adapters mediating the interaction among two services with different interfaces, and it is implemented in Websphere environment. Our model addresses similar problems, but it is more general than that. In fact, while [16] concerns the mediation between two services, our proposal addresses the mediation among several services taking part in multi-party interaction: the choreography.

## References

1. Peltz, C.: Web Services orchestration and choreography. *Innovative Technology for Computing Professionals* **36**(10) (2003) 46–52
2. W3C: Web Services Definition Language, <http://www.w3.org/TR/wsdl> (2002)
3. Lu, R., Sadiq, S.: A survey of comparative business process modeling approaches. In: *Proc. of 10th Int. Conf. on Business Information Systems (BIS)*, LNCS 4439, Poznan, Poland (2007) 82–94
4. van der Aalst, W., van Hee, K.: *Workflow Management - Models, Methods, and Systems*. The MIT Press (2002)
5. Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard, D., Shewchuk, J., Storey, T.: *Web Services Coordination (WS-Coordination)*, <http://www-106.ibm.com/developerworks/library/ws-coor/> (2002)
6. GigaSpaces: GigaSpaces SBA, [http://www.gigaspaces.com/pr\\_overview.html](http://www.gigaspaces.com/pr_overview.html) (2008)
7. Ahuja, S., Carriero, N., Gelernter, D.: Linda and friends. *IEEE Computer* **19**(8) (1986) 26–34
8. W3C: Web Services Choreography Description Language version 1.0, <http://www.w3.org/TR/ws-cdl-10/> (2005)
9. W3C: Web Services Conversation Language (WSCL), <http://www.w3.org/TR/wscl10> (2002)
10. Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S.: *Web Service Choreography Interface 1.0*, <http://ifr.sap.com/wsci/specification/wsci-specp10.html> (2002)
11. Ardissono, L., Petrone, G., Segnan, M.: A conversational approach to the interaction with Web Services. *Computational Intelligence* **20**(4) (2004) 693–709
12. DERI International: Web Service Modeling Ontology, <http://www.w3.org/Submission/WSMO/> (2005)
13. DERI International: Web Service Modelling eXecution environment, <http://www.wsmx.org/> (2006)
14. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.M., Toumani, F.: Developing adapters for Web Services integration. In: *Proc. 17th Conf. on Advanced Information Systems Engineering (CAiSE'05)*, Porto, Portugal (2005)
15. Snoeck, M., Lemahieu, W., Goethals, F., Dedene, G., Vandenbulcke, J.: Events as atomic contracts for component integration. *Data knowledge & knowledge engineering* **51** (2004) 81–107

16. Motahari Nezhad, H., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: Proc. of 16th Int. World Wide Web Conference (WWW'2007), Banff, CA (2007) 993–1002