

## Chapter 8

# ENABLING FLEXIBLE INTERACTION WITH WEB SERVICES

Liliana Ardissono, Giovanna Petrone and Marino Segnan

*Dipartimento di Informatica, Universita' di Torino*

{liliana, giovanna, marino}@di.unito.it

**Abstract** The exploitation of Web Services in e-commerce is affected by their limited support for the interaction with the service consumers. The current standards for the publication of services fail to specify the conversation flow for the invocation of the operations; thus, only simple requests can be performed by the consumers.

In this chapter, we present a conversation model supporting the management of long-lasting interactions between service provider and consumer, where several messages have to be exchanged before the service is completed. Other approaches assume that both the service producer and the consumer locally manage the contextual information to carry out the conversation with the partner. In contrast, our model supports the consumers during the service invocation in order to simplify the establishment of short-term business relationships. Our approach is based on an interaction model derived by simplifying Speech-Act Theory. Moreover, the management of the conversation turns is based on the communication techniques proposed by the emerging standards for Web Services.

## 1. Introduction

Web Services are subject to several limitations that reduce their applicability to realistic domains. For instance, the emerging service publication standards, such as WSDL (Web Services Description Language, (W3C, 2002b)), support the specification of the static interfaces of elementary services. However, these standards do not enable the service provider to specify the order of the operations to be invoked by the consumer during service execution. Therefore, the only services that can be

successfully provided are those requiring one-shot interactions with the consumers. Moreover, these standards support the invocation of operations characterized by very specific signatures with fixed parameter lists. Although this is not a problem when the requirements for the service can be specified by the consumer in a pre-determined way, it challenges the provision of highly interactive services.

At the same time, current work on workflow management is focused on service composition in the Web, but the proposed approaches assume a very simple type of interaction with the suppliers whose services have to be invoked. For instance, BPEL4WS (Business Process Execution Language for Web Services, (Curbera et al., 2002; Curbera et al., 2003)) supports complex service compositions. However, the management of the interaction only deals with low level communication issues such as transaction management; see (Cabrera et al., 2002).

In order to make service execution possible even when the involved suppliers require complex interactions, service invocation has to be modeled as a long-lasting conversation where several messages are exchanged before the service is completed; e.g., requirements acquisition, negotiation and other types of interaction. For instance, during the interaction with a Web Service configuring medium complexity items, the selection of the item features may require more than one step. Moreover, failures may occur and have to be repaired before the solution for the consumer is generated. Finally, in some cases, the Web Service may require suspending the interaction, e.g., waiting for a sub-supplier or a human operator to contribute to the generation of the solution. At the same time, the consumer should be able to match its own business logic to the provider's logic. For instance, not only the provider, but also the consumer might need to suspend the interaction because it needs supplementary information from the customer before choosing certain product features.

Note that a loosely coupled approach to the management of the interaction is needed to support successful business interactions and, at the same time, enable the consumers to match the provider's conversation requirements to their own business logic. Moreover, the communication capabilities of service providers and consumers should be enhanced by means of a lightweight approach. This is particularly important for the consumers, which should be able to start e-business interactions with heterogeneous providers, also outside well established Business-to-Business relationships.

In this chapter, we present a framework supporting the management of long-lasting interactions between Web Services providers and consumers. Our aim was to enable the conversation participants to dynamically

select the messages to be exchanged during the service execution and to suspend and resume the interaction according to their needs. Thus, we have adopted a conversational approach to the management of the interaction. In particular, we have taken the Speech-Act Theory model of dialog management (Searle, 1975; Cohen and Levesque, 1990) as a starting point for the development of our conversation framework, which has then been simplified in order to support its integration with the emerging standards for the publication of Web Services.

The rest of this chapter is organized as follows: Section 2 positions our proposal in the current research about Web Services. Section 3 describes a representation of conversation flow based on Speech Acts. Section 4 presents our approach to the management of conversations between service consumers and providers. Section 5 describes the framework architecture and implementation. Section 6 compares our proposal to the related work and Section 7 concludes the chapter.

## **2. Motivations**

As the management of long-lasting interactions is recognized as a central issue for the provision of realistic services as Web Services, some XML-based standards for the specification of the conversation flow in e-business interactions with Web Services have been recently proposed and are submitted as W3C standards. Current approaches support the specification of complex conversations. For instance, WSCL (Web Services Conversation Language (W3C, 2002a)) and WSCI (Web Services Choreography Interface (Arkin et al., 2002)) introduce an explicit representation of Web Services interaction processes, aimed at defining the admissible sequences of messages to be exchanged. For this purpose, WSCL exploits a sequence diagram model that the service provider and the consumer should interpret to handle the conversation, while WSCI introduces the notion of interaction process, with the definition of timing constraints on the service invocation. Moreover, cpXML (IBM's Conversation Support (Hanson et al., 2002b)) introduces an explicit notion of Conversational Policy as a machine readable specification of a pattern of message exchange in a conversation, which can be used to steer the interaction with complex Web Services at the consumer application side.

Unfortunately, these approaches assume that the provider and the consumer separately maintain an internal record of the conversation state and that they autonomously manage the turn-taking activity and the selection of the next conversation action to be performed. This assumption is problematic because, in order to guarantee that the consumer has the required interaction capabilities, a conversation automa-

ton compatible with the one defined by the Service provider has to be provided at the consumer side.

In order to support loosely-coupled communication between the service provider and its consumers, the consumer applications should be assisted in the invocation of operations. This means that only the service provider should be responsible for maintaining the state of the conversation and identifying the eligible continuations. A flexible, but seamless conversation model is needed to support the dynamic invocation of Web Services and the present chapter contributes to the definition of this model. Before presenting our approach, we discuss the following issues essential to the identification of our contribution.

- We assume that the matching phase between service description and request has been performed and we focus on the service execution phase. It should be noted that we leave out the matching phase because it represents an important task deserving separate treatment. On the one hand, the identification of the service provider can be seen as a separate activity, to be performed either directly, or by exploiting mediation agents; e.g., see (Klusck and Sycara, 2001). On the other hand, after a provider is identified, an explicit and possibly complex binding activity has to be carried out by the consumer in order to associate the operations to its own business logic. This activity may require the intervention of a human administrator, who has to carefully analyze the meaning of the operations and their arguments, possibly requiring the sharing of the underlying domain ontology with the Web Service.
- The management of conversations should not be confused with the service composition that, in the Web Services research, is typically handled by workflow engines, such as BPEL4WS. In fact, these are complementary to one another. In particular, the conversation management enriches the flexibility in the invocation of the individual suppliers whose services are composed by the consumer.

### **3. A Speech-Act Based Representation of Conversations**

Social behavior of human and software agents has been traditionally described by exploiting the notion of Speech Act as a way of performing actions by exploiting language; see (Austin, 1962) and (Searle, 1975) for details. In particular, communicative behavior in task-oriented interactions has sometimes been described by means of finite state automata defining the admissible sequences of speech acts to be executed (Stein and Maier, 1994). Moreover, hierarchical scripts and plan-based ap-

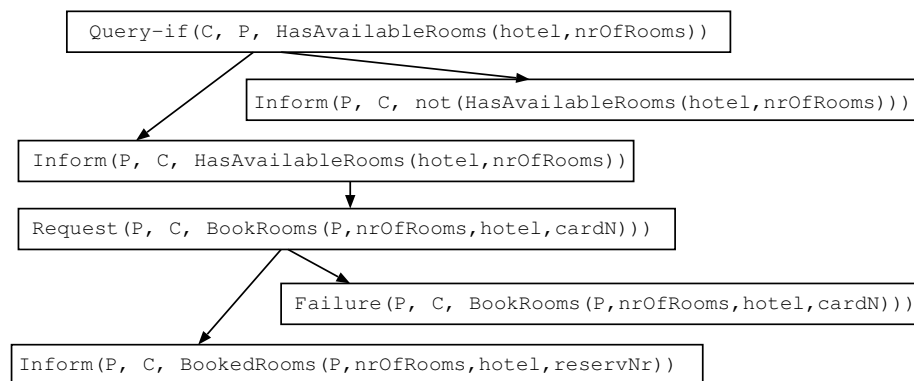


Figure 8.1. Speech-Act based interaction flow for the hotel booking service.

proaches have been introduced to model goal-oriented behavior (Cohen and Levesque, 1990) and to separate the conversational behavior from the domain-dependent activities carried out by the agents (Carberry et al., 1999; Rich et al., 2002). Roughly speaking, the agents cooperating at a domain-level activity communicate with each other to coordinate their own internal processes and to verify the feasibility of the actions they have to perform.

Web Services and their consumers can be modeled as interacting agents and similar approaches can be used to model conversational behavior. In particular, the messages exchanged during the service execution can be seen as conversational actions performed to carry out a task-oriented dialog between service consumer and provider. For the specification of the conversation flow, the roles of the participants have to be defined. Although multi-party conversations may be specified, in the following we concentrate on conversations between two participants and we define the Consumer and the Service Provider roles (multi-party conversations can be handled in a straightforward way by defining additional roles).

As a concrete example, we consider a simple hotel booking service. Figure 8.1 shows a possible representation of the admissible turn sequences in this service. Each action specifies a speech act representing a conversational turn performed by one of the participants. We have named the speech acts according to the FIPA specifications; see (FIPA, 2000).<sup>1</sup> The first argument of a speech act represents the role filler that should perform the act, i.e., the agent sending the message. The second argument denotes the recipient and the third one represents the content of the speech act. The arcs represent the possible turn sequences and the

actions having more than one output arc are mutually exclusive. The interaction starts with the *Query-if* action, where the consumer  $C$  asks the service provider  $P$  whether the hotel has at least  $nrOfRooms$  available rooms ( $HasAvailableRooms(hotel, nrOfRooms)$ ). The service provider may answer positively ( $Inform(P, C, HasAvailableRooms(hotel, nrOfRooms))$ ), or negatively ( $Inform(P, C, not(HasAvailableRooms(hotel, nrOfRooms)))$ ). In the second case, the interaction has to be terminated. In the first case, the consumer may request to book a certain number of rooms by providing a credit card number. The service provider may either provide the consumer with the reservation number or notify that the booking action has failed.

The interaction flow specified in Figure 8.1 is simpler than the hierarchical plans applied to manage human-computer dialog; see (Carberry et al., 1999). In fact, the types of interaction to be modeled are much simpler and predictable. At the same time, we want to minimize the amount of information that has to be understood by the consumer. For instance, we have not specified any preconditions on the conversation turns. Moreover, each participant is not informed about the internal state of its party. Although these two types of information would help to predict the actions that will be performed by the other participant, the only type of information to be understood by the interactants is the correct sequence of speech acts that can be executed.

The publication of the conversation script enables a consumer to request the services provided by another agent by following a specific interaction flow. However, this approach is not applicable to the current situation of Web Services, for at least two reasons. First, the imposition of speech acts on Web Services, now publishing services by means of very simple languages such as RPC invocations or WSDL operations, is not realistic. Second, the identification of the admissible reactions to the provider's messages requires that the consumer keeps an internal representation of the interaction context which includes, at minimum, the current focus of the conversation, i.e., the most recently executed action. These requirements are not desirable because they impose supplementary efforts on the consumer than those required to invoke standard Web Services.

#### 4. Server-Side Conversation Management

In order to support the management of lightweight and loosely coupled conversations, we propose to make the management of the interaction easier for the consumer, charging the service provider with the control of the service invocation. More specifically, we propose that:

- The service provider publishes the services by specifying the WSDL operations to be invoked. This enables the consumers to bind the invocations of operations to their own business logic.
- The specification of the interaction flow is based on a flexible, but simple representation formalism supporting the definition of the correct sequence of turns to be exchanged without the overhead of the pure speech-act model.
- The service provider maintains a local interaction context, for each active conversation, to guarantee that at least one of the participants controls the conversation.
- To make the consumer aware about the eligible actions it may perform, at each step, the provider enriches the messages it sends to the consumer with contextual and turn management information. The contextual information may be void in trivial interactions. The turn management information consists of the eligible actions (henceforth, next operations) that the consumer may perform to carry the interaction one step forward.

#### 4.1 Conversation Flow Specification

The specification of the interaction flow can be simplified by modeling the interaction turns as generic conversational activities where the performed speech act (*Inform*, *Query-if*, etc.) is omitted. Each conversational action represents a simplified speech act (Searle, 1975), where the sender asks the recipient to perform the operation specified as an argument. The conversational actions have the following arguments:

- The message sender, which may be either the consumer  $C$ , or the service provider  $P$ .
- The recipient of the message (similar).
- The requested operation, i.e., the operation that the sender invokes on the recipient by performing the speech act. The actor of the requested operation may be omitted because it coincides with the recipient of the message.
- The list of the possible continuations of the conversation (*nextOps*). As the service provider has the control of the interaction, this argument is only present in the messages to be received by the consumer. The argument includes the set of operations offered by the provider which the consumer may invoke in the next conversation step.

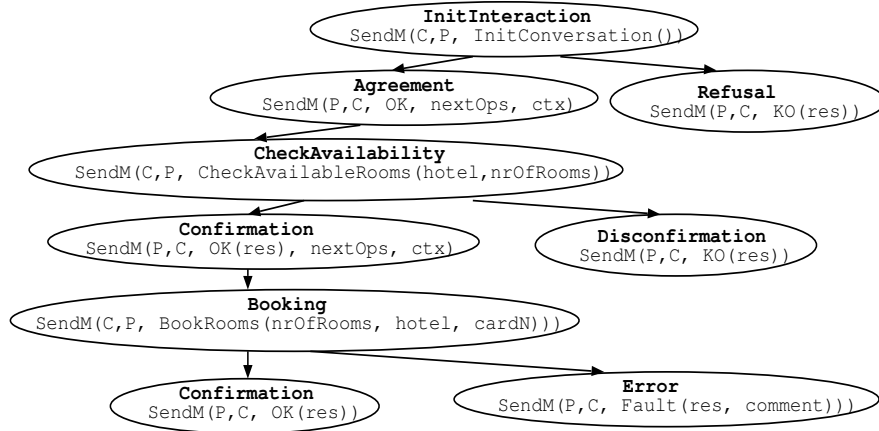


Figure 8.2. Complete conversation flow for the hotel booking service.

- A context argument, storing information about the state of the interaction (*ctx*). Similar to the *nextOps* argument, the *ctx* one is only present in the messages directed to the consumer.

The structure of the context argument depends on the application domain. The context object can be empty in simple and deterministic interactions, where the consumer has at most one conversation turn to choose from, i.e., the next operation argument of the includes at most one element. However, we introduced the context argument in the messages to support the development of consumers displaying different levels of initiative during the interaction with the Web Service. Although we cannot make any assumptions on the consumer's decision capabilities, the binding activity in a complex consumer might concern the definition of conditions on the invocation of the operations on the service provider. For instance, the consumer might wish to inhibit the invocation of some operations before the interaction reaches a certain fulfillment state, to be specified in the contextual information.

Figure 8.2 shows the simplified representation of the interaction flow in our hotel booking service. In the figure, the nodes represent conversation turns and the arrows define the possible turn sequences. The nodes having more than one output arrow represent alternative conversational actions. As shown in the figure, an interaction is triggered by a consumer that sends a message to request the *InitConversation* action specified at the root node of the conversation script. This message starts the hand-shake between the participants and the provider may accept or refuse the conversation (see the successor nodes in the

script), sending a notification message. In the positive case, the conversation is carried out as specified in the script, otherwise it is terminated. Each conversation turn is represented as a send message (*SendM*) activity specifying the related arguments (the sender of the message, the recipient, etc.). For instance, *Query-if(C, P, HasAvailableRooms(hotel, nrOfRooms))* in Figure 8.1 corresponds to *SendM(C, P, CheckAvailableRooms(hotel, nrOfRooms))*. Moreover, *Inform(P, C, HasAvailableRooms(hotel, nrOfRooms))* is replaced by *SendM(P, C, OK(res), nextOps, ctx)*.

Two kinds of operations may be the arguments of a conversation turn:

- *Domain-level operations*, such as *CheckAvailableRooms* and *BookRooms*, representing domain dependent operations to be invoked during the service execution.
- *Communicative actions*, such as *InitConversation*, *OK*, *Fault*, *Suspend*, *Resume* and *ReceiveResult*, that are domain-independent and have to be invoked during the service execution to coordinate the behavior of the service provider and consumer.

The communicative actions have to be offered by both the provider and the consumer. The domain-level operations are only offered by the service provider because they are concerned with the service execution. For instance, the *Suspend* and *Resume* actions, which can be invoked on the provider as well as on the consumer, are aimed at suspending a conversation and resuming it later on. Moreover, the *ReceiveResult* action has to be implemented by the consumer in order to collect the results of the operations performed by the provider. In the figure, the domain-level operations are show in boldface.

## 4.2 Interaction Management

The conversation flow specification enables a consumer to bind the invocation of the service operations to its own business logic, by associating the invocation of operations to its own internal processes. However, the exhaustive specification of the conversation flow may be challenging for highly interactive Web Services. For instance, consider a Web Service selling configurable items. The operations to be performed during the configuration process can be clearly defined, e.g., selecting the needed components, setting values for the features of the components, and so forth. However, the features whose values have to be set at each step cannot be *a priori* determined because they depend on the components required by the consumer and on the inferences performed by the configuration system employed by the Web Service. Therefore, the

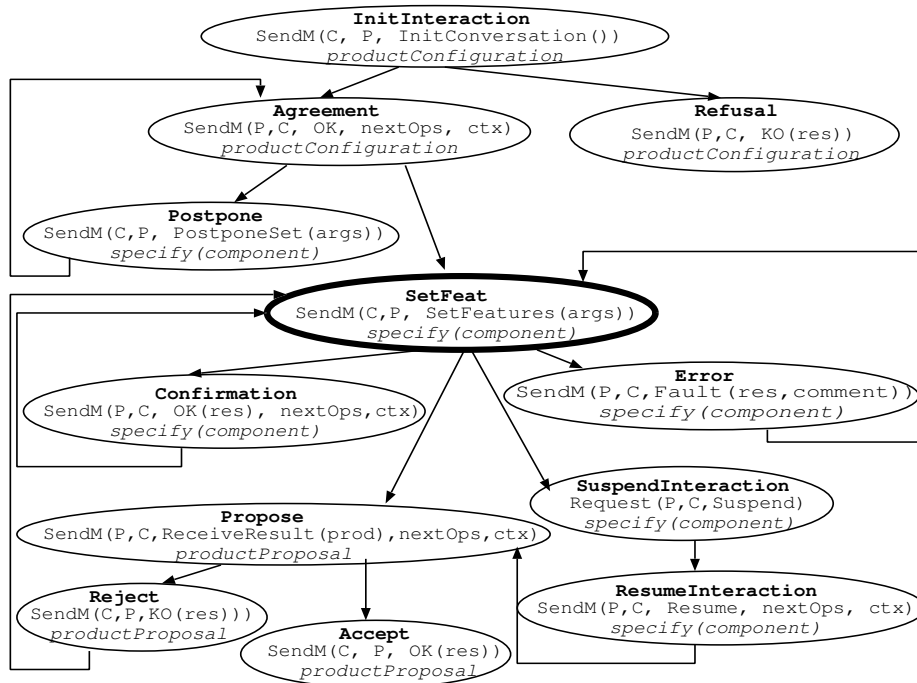


Figure 8.3. A portion of the conversation script of the Configuration Web Service.

Web Service has to dynamically determine the operations that can be invoked during the exploration of the search space by taking contextual information about the interaction into account.

The provider can enforce the run time interaction management by instructing the consumer about the operations to be invoked and their actual parameters. This is possible if, for each active interaction, the provider:

- Maintains a context object specifying contextual information, such as the current focus of the interaction.
- Exploits the conversation flow specification to identify, at each step, the possible continuations of the interaction.

Given the script representing the conversation flow, the current focus is the node corresponding to the last speech act performed either by the service provider or by the consumer; see (Carberry et al., 1999).

When a consumer starts a conversation with the provider, the current focus is set to the root node of the script (*InitInteraction*). The provider then moves the current focus in the script according to the messages it

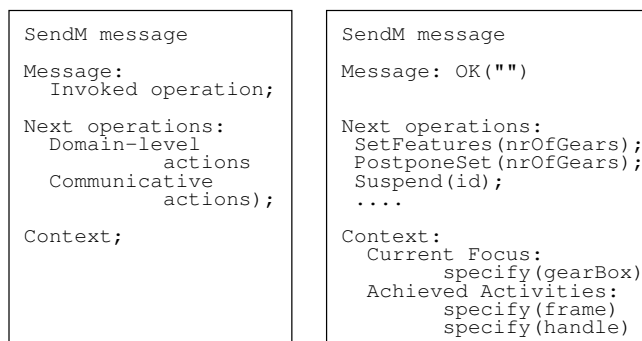


Figure 8.4. Format of messages exchanged between service provider and consumer.

sends or receives. When the consumer receives a message, it executes the requested operation. It then extracts the eligible continuations and selects the one to perform. Note that each turn is an asynchronous message that one of the participants performs to carry the conversation one step forward. If the message does not reach the recipient, the interaction is suspended, with a time out.

### 4.3 Supporting the Consumer-Side Participation in the Interaction

In the following, we describe our framework for the management of lightweight conversations between service consumers and providers. Figure 8.3 shows the script representing the conversation flow of a Web Service supporting the configuration of medium complexity products, such as bicycles. As shown in the figure, this service requires complex interactions between the participants.<sup>2</sup> For instance, the feature setting operation has to be invoked by the consumer several times, until the item is completely configured. When the consumer invokes the *SetFeatures* operation (*SetFeat* node) to specify a set of product features, such as the number of gears, the provider may respond in different ways. For instance, it may confirm that the configuration step was successful (*Confirmation* node) and enable another invocation of the *SetFeatures* operation to carry the configuration one step forward. Alternatively, the service provider may notify the consumer about a failure in the configuration (*Failure*) and it may enable the selection of other values for the conflicting features. In the conversation script, the *SetFeatures* operation has a formal parameter (*args*). This parameter has to be bound to the actual list of features set at each interaction step, depending on the evolution of the configuration process.

Figure 8.4 sketches the format of the messages sent by the provider to the consumer during the configuration of a bicycle.

- On the left side, the figure shows the abstract structure of the *SendM* messages. In the depicted message structure, the sender and receiver arguments are omitted because they are specified in the message header. More specifically, the messages representing the conversation turns can be seamlessly extended with turn-management information. Being implemented as SOAP messages, this information can be added by including new parts within the message body (W3C, 2001).
- On the right side, a sample instance message is shown. In the message, the provider sends a positive acknowledgment (*OK*) representing the successful execution of a previously invoked operation. The provider also specifies that the consumer may invoke *SetFeatures* to set the number of gears the consumer needs, *PostponeSet* to postpone the setting to a later stage of the interaction, or it may suspend the conversation.

The execution of these operations is aimed at carrying out the configuration of the gear box (*Current Focus*); moreover, the service has already configured the frame and the handle of the bicycle (*Achieved Activities*).

The sample context object shown at the right side of Figure 8.4 sketches a possible representation of the completion state of service offered by our Configuration Web Service. During the interaction with a consumer, the context is enriched to show that new components of the bicycle have been successfully configured. In this situation, the consumer could exploit the contextual information to select the most appropriate continuation of the interaction in an informed way, in contrast to a naive consumer. For instance, the informed consumer might wish to configure the brakes of the bicycle only after the gears have been configured. To achieve this behavior, when the consumer binds the service invocation to its own business logic, it should condition the invocation of *SetFeatures(brakes)* to the fact that *specify(gears)* is included in the context as an achieved activity. Of course, this type of behavior is possible only if the following conditions are satisfied: the consumer manages the contextual information on its own and is able to subordinate the invocation of operations to the satisfaction of conditions on the context state. Moreover, the Configuration Web Service offers a *PostponeSet* operation enabling the consumer to postpone configuration decisions; this is important to reconcile the business logics of the two participants.

```

<types>
<schema targetNamespace="http://example.com/configBicycle.xsd"
        xmlns="http://www.w3.org/2001/XMLSchema">

    <element name="SetFeatures" type="SetFeatAct"/>
    <complexType name="SetFeatAct">
        <element name="feature" type="Feature" minOccurs="0" maxOccurs="unbounded"/>
    </complexType>
    <complexType name="Feature">
        <element name="featureName" type="string"/>
        <element name="featureValue" type="Domain"/>
        <element name="featureDomain" type="Domain"/>
    </complexType>
    <complexType name="Domain">
        <choice>
            <element name="IntInterval" type="IntIntervalDomain"/>
            <element name="Boolean" type="BooleanDomain"/>
            ...
        </choice>
    </complexType>
    <complexType name="IntIntervalDomain">
        <all>
            <element name="upperBound" type="integer"/>
            <element name="lowerBound" type="integer"/>
        </all>
    </complexType>
    ...
</schema>
</types>

```

Figure 8.5. XML-schema definition of the operations and of their arguments.

## 4.4 Representing the Conversation Flow in XML

A standard representation formalism is needed to publish the conversation scripts on the Web. We thus decided to exploit an XML dialect, which we defined by taking inspiration from the IBM's WSFL (Web Services Flow Language, (IBM, 2002)) service composition language. As described in (Ardissono et al., 2003) our representation enables the specification of WSDL operations and the partial order relations between their invocation.

Figure 8.5 shows a portion of the XML-schema defining the structure (operation-name and arguments) of the operations to be executed during the configuration of a bicycle. For instance, the *SetFeatures* operation has an argument containing a list of *Feature*, which is a complex type composed of *featureName*, *featureValue* and *featureDomain*. The *featureDomain* element specifies the set of admissible values that the feature can take. For example, *Domain* can be an *IntIntervalDomain* with its upper and lower boundaries, a *BooleanDomain*, a *StringEnum*

```

<flowModel name="ConfigurationFlow" serviceProviderType="ConfigurationService">

  <serviceProvider name="configServer" type="ConfigurationServiceWS">
    <locator type="static" service="ourConfigServer.com"/>
  </serviceProvider>
  <consumer name="consumer" type=""/>

  <activity name="sendFeaturesSpecification">
    <performedBy actor="consumer"/>
    <implement>
      <export><target portType="configServicePT" operation="SetFeat"/></export>
    </implement>
  </activity>

  <activity name="sendFeaturesConfirmation">
    <performedBy actor="configServer"/>
    <implement>
      <export>
        <target portType="configServicePT" operation="Confirmation"/> </export>
      </implement>
    </activity>

  <activity name="sendErrorMessage">
    <performedBy actor="configServer"/>
    <implement>
      <export> <target portType="configServicePT" operation="Error"/> </export>
    </implement>
  </activity>
  ...

  <controllink source="sendFeaturesSpecification" target="sendFeaturesConfirmation"/>
  <controllink source="sendFeaturesSpecification" target="sendErrorMessage"/>
  <controllink source="sendFeaturesConfirmation" target="sendFeatureSpecification"/>
  <controllink source="sendErrorMessage" target="sendFeatureSpecification"/>
  ...
</flowModel>

```

Figure 8.6. Portion of the conversation flow for our Configuration Web Service.

merationDomain, and so forth. Note that the *featureValue* element, representing the current value of the feature, has *Domain* type; this is due to the fact that during the configuration process its original domain could be progressively restricted, until the final value is selected.

For each operation, a WSDL declaration specifies binding and ports information. For brevity, we omit the complete WSDL specification of the operations, whose identifiers are reported as boldface labels in the nodes of the conversation script. For instance, the *SetFeatures* operation in Figure 8.5 is embedded in the *SetFeat* WSDL operation. Figure 8.6 shows a portion of the flow model for the Configuration Web Service:

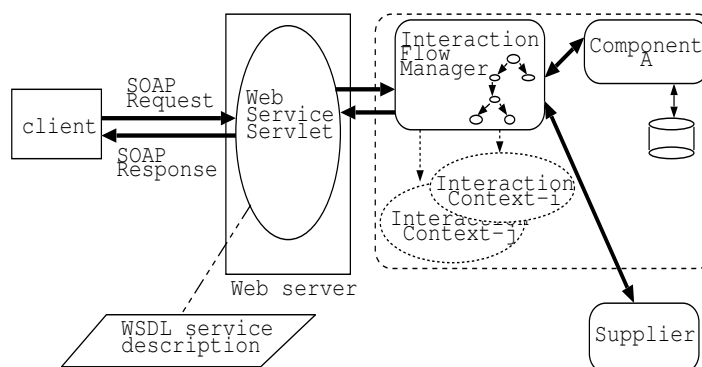


Figure 8.7. Conversation framework.

- The *serviceProvider* and *consumer* elements characterize the two conversational roles and the peers filling the roles.
- Each activity to be carried out is defined by specifying the requested WSDL operation and the actor that should perform the activity. For instance, activity *sendFeatureSpecification* has to be performed by the consumer and has as its content the *SetFeat* WSDL operation.
- The *controlLink* elements define the partial order relations between activities.

The activities represent the conversation turns and are performed by the message sender to trigger the execution of the operations specified as their arguments on the receiver.

## 5. Framework Architecture and Implementation

In order to manage the conversation, the service provider and consumer should run, respectively, a Conversation Manager and a Conversation Client module; Figure 8.7 sketches the architecture of the proposed framework.

- The Conversation Manager would exploit the conversation script (see Figure 8.3) to control the service provider's communicative behavior. For each interaction session, the Conversation Manager of the provider maintains the active state of the interaction as a description of the contextual information concerning the conversation. This information is needed to compute the next operations available to the consumer and to support other types of behav-

ior, such as the suspension of a conversation and the subsequent restart.

- The Conversation Client would parse the incoming messages and send back the responses. More specifically, it should facilitate the reception and interpretation of messages at the consumer side (by extracting the next operations and their actual parameters). Moreover, the Client should support the invocation of operations by performing generating and delivering the outbound messages to the provider.

We are developing a set of Java libraries which facilitate the development of a Conversation Manager and a Conversation Client modules supporting the communication between Web Service providers and their consumers.

In particular, a Java-based Conversation Manager module enables the Web Service provider to keep track of the asynchronous communication with the interacting consumer applications. The proposed architecture of the Web Service Provider is shown in Figure 8.7. A Servlet supports the (SOAP) HTTP-based communication with the consumer, by catching the incoming requests and forwarding them to the Conversation Manager for their management. Indeed, the Conversation Manager is the core of the Servlet listening to the incoming requests, invoking the appropriate components to execute the services and sending the SOAP response messages to the consumer applications. Depending on the binding to the Web Service provider's business logic, this module can invoke different types of components to provide the service. For instance, a service could rely on an internal component, such as Component A in Figure 8.7, or on an external Supplier. The Conversation Manager may execute a conversation flow automaton for the management of the interaction sessions with the consumers in order to compute the possible continuations of each interaction. Although an infrastructure supporting the definition of general purpose conversation automata is not yet available, we have developed a prototype Conversation Manager that implements the script shown in Figure 8.3 and that can be easily customized to satisfy the interaction requirements of different application domains.

Our framework also supports the consumer by offering a Java-based Conversation Client that may be downloaded and run in order to handle the interaction with a Web Service. Similar to the Conversation Manager, the execution of the Client has the prerequisite that the consumer binds the invocation of the operations to its own business logic.

## **6. Related Work**

The Semantic Web community (Web Services Coalition, 2002) is defining standards for the publication of Web Services aimed at overcoming the main limitations of WSDL and the ones of the emerging workflow management standards. The semantic approach differs from the pure XML-based ones because it specifies the domain ontology underlying the service, the meaning of the operations to be invoked and the service choreography. The main advantage is that the service offered by a provider may be unambiguously understood by the (UDDI) registries, therefore enhancing their capability to redirect consumers to the most suitable providers; see (Kamamura et al., 2003). In order to facilitate the interoperability between service consumers and providers, some recent work also proposes prototype infrastructures for the runtime conversation management. These infrastructures rely on the semantic representation of the services to guide the interaction, e.g., by instructing the consumer during the service invocation (Paolucci et al., 2003).

Although the semantic Web approach is a promising solution to the interoperability in the internet, the current proposals are too complex to be applied in real-world examples. For instance, these approaches rely on sophisticated representations of the services to be invoked; although translation tools assist the binding to the consumers' business logics, this remains a rather complex process. Moreover, the selection of the operations to be invoked, depending on the service choreography, is based on the exploitation of inference engines, such as rule-based ones, imposing relevant overhead on the management of the interaction. In our work we are deeply concerned with the scalability and applicability requirements of the Web. For this reason, we try to reduce the complexity imposed by semantic information as much as possible, and to handle the interaction between service consumer and provider in a lightweight way, at least at the consumer side.

The same considerations are useful to relate our work to the previous Multi-Agent Systems research, which regulated the interaction between agents by defining coordination protocols such as the FIPA Contract Net (FIPA, 2000) and JAFMAS conversations (Chauhan, 1997). Unfortunately, the application of these approaches to the current Web Services is not straightforward because it requires agreement on non-standard communication languages and interaction protocols, as done in the AgentCities project (Agentcities, 2002). Moreover, these approaches assume that each participant in the conversation is able to adhere to the shared

conversation protocol and this is a strong requirement for Web Service consumers.

Our interaction model differs from the other XML-based proposals for the publication of Web Services, such as WSCL (W3C, 2002a), WSCI (Arkin et al., 2002) and cpXML (Hanson et al., 2002b), because they assume that each conversation participant separately maintains its own internal record of the conversation state. In contrast, we propose that only the service provider handles the state of the conversation and that the consumer application is assisted in the service invocation. Moreover, WSCL and WSCI conform to WSDL in the definition of *request-response* and *solicit-response* operations; thus, they cannot support a fine-grained specification of the conversation turns.

Finally, as far as cpXML is concerned, the consumer and provider roles require the same effort from the conversational point of view. In contrast, we try to simplify the implementation of the consumer by supplying it with the minimum amount of information needed to interact with the provider. At the same time, our approach does not impose extra overhead on the service provider, for two reasons. First, the provider has to maintain the context for each interaction session, otherwise, it would not be able to correctly execute the service requests. Second, the provider knows the details of the execution of its own services. Therefore, it may guide the consumers during the management of complex interactions.

Both cpXML and our work aim at decoupling the service provider and consumer's business logics by mediating their interaction by means of the conversational activity. However, our model has the potential to separate the consumer from the provider in a clearer way because it does not require the execution of a particular conversation policy.

## 7. Discussion

We have presented a conversation model supporting the management of long-lasting interactions between Web Service providers and consumers. Our model is based on the idea that the service provider should support the consumers in the service invocation, in order to guarantee that the operations are requested in the correct order and that their parameters are suitably instantiated by the consumer. These characteristics make our model particularly suited to the management of highly interactive Web Services, such as those supporting the customization of complex products and services, which define the eligible sequence of invocations at run time, depending on the evolution of the configuration process.

As described in (Ardissono et al., 2003), our proposal builds on the Speech Acts Theory (Searle, 1975) that represents communicative behavior as actions performed by an actor towards a recipient. Our representation does not support the specification of different types of Speech Acts, with their semantics (FIPA, 2000). However, it clearly separates the conversational activity from the domain-specific behavior that is represented as object level actions the partners “talk about” during the interaction. The explicit representation of the conversation participants and of their social behavior supports the detailed and unambiguous specification of the possible sequences of interaction turns, at the granularity level of the individual messages to be exchanged. Moreover, the model is simplified in several aspects in order to take into account the applicability requirements that can seriously affect the usefulness of the model in real cases. Finally, our model clearly separates the issues concerning the internal implementation of a Web Service from the communication protocol defining the invocation of its operations. As noted in (W3C, 2002a) and (Hanson et al., 2002a), no knowledge about the implementation of the service providers should be needed to invoke them.

## Notes

1. FIPA, the Foundation for Intelligent Physical Agents, is an international organization aimed at producing software standards for heterogeneous and interacting agents and agent-based systems. More information about the organization can be found in (FIPA, 2000).

2. Sharing the ontology representing the product structure is necessary to let the consumer understand the individual product features to be set.



## References

- Agentcities (2002). Agentcities network services. <http://www.agentcities.net/>.
- Ardissono, L., Goy, A., and Petrone, G. (2003). Enabling Conversations with Web Services. In *Proceedings of 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 819–826, Melbourne, Australia.
- Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacs-Nagy, P., Trickovic, I., and Zimek, S. (2002). Web Service Choreography Interface 1.0. <http://ifr.sap.com/wsci/specification/wsci-specp10.html>.
- Austin, J.L. (1962). *How to Do Things with Words*. Harvard University Press, Cambridge, Mass.
- Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard, D., Shewchuk, J., and Storey, T. (2002). Web Services Coordination (WS-Coordination). <http://www-106.ibm.com/developerworks/library/ws-coor/>.
- Carberry, S., Chu-Carroll, J., and Elzer, S. (1999). Constructing and Utilizing a Model of User Preferences in Collaborative Consultation Dialogues. *Computational Intelligence*, 15(3):185–217.
- Chauhan, D. (1997). *JAFMAS: A Java-Based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, University of Cincinnati, Cincinnati, OH.
- Cohen, P.R. and Levesque, H.J. (1990). Rational Interaction as the Basis for Communication. In Cohen, P.R., Morgan, J., and Pollack, M.E., editors, *Intentions in Communication*, pages 221–255. MIT Press.
- Curbera, F., Golland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002). Business Process Execution Language for Web Services, Version 1.0. <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>.
- Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. (2003). The Next Step in Web Services. *Communications of the ACM, Special Issue on Service-Oriented Computing*, 46(10).
- FIPA (2000). Foundation for Physical Intelligent Agents. <http://www.fipa.org/>.

- Hanson, J.E., Nandi, P., and Kumaran, S. (2002a). Conversation Support for Business Process Integration. In *Proceedings of IEEE International Enterprise Distributed Object Computing Conference (EDOC-02)*, pages 65–74, Lausanne, Switzerland.
- Hanson, J.E., Nandi, P., and Levine, D. (2002b). Conversation-Enabled Web Services for Agents and E-Business. In *Proceedings of International Conference on Internet Computing (IC-02)*, pages 791–796, Las Vegas, Nevada.
- IBM (2002). Web Services Flow Language. <http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>.
- Kamamura, T., DeBlasio, J., Hasegawa, T., Paolucci, M., and Sycara, K. (2003). Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry. In *Proceedings of International Conference on Service-Oriented Computing (ICSOC 2003)*, pages 208–224, Trento, Italy.
- Klusch, M. and Sycara, K. (2001). Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In Omicini, A., Zambonelli, F., Klusch, M., and Tolksdorf, R., editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 8, pages 197–224. Springer-Verlag.
- Paolucci, M., Sycara, K., Nishimura, T., and Srinivasan, N. (2003). Toward a Semantic Web E-commerce. In *Proceedings of 6th International Conference on Business Information Systems (BIS'2003)*, Colorado Springs, Colorado.
- Rich, C., Lesh, N., Rickel, J., and Garland, A. (2002). A Plug-in Architecture for Generating Collaborative Agent Responses. In *Proceedings of 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 782–789, Bologna, Italy.
- Searle, J.R. (1975). Indirect Speech Acts. In Cole, P. and Morgan, J., editors, *Syntax and Semantics: Speech Acts*, volume 3, pages 59–82. Academic Press, New York.
- Stein, A. and Maier, E. (1994). Structuring Collaborative Information-Seeking Dialogues. *Knowledge-Based Systems*, 8(2-3):82–93.
- W3C (2001). Simple Object Access Protocol (SOAP) 1.2. <http://www.w3.org/TR/2001/WD-soap12-20010709/>.
- W3C (2003a). Web Services Conversation Language (WSCL). <http://www.w3.org/TR/wscl10>.
- W3C (2003b). Web Services Definition Language. <http://www.w3.org/TR/wsdl>.
- Web Services Coalition (2002). DAML-S: Web Service Description for the Semantic Web. In *Proceedings of International Semantic Web Conference*, pages 348–363, Chia Laguna, Italy.