Enabling Conversations with Web Services

L. Ardissono, A. Goy and G. Petrone
Dipartimento di Informatica - University of Torino
Corso Svizzera 185 - Torino, Italy
{liliana, goy, giovanna}@di.unito.it

ABSTRACT

The emerging standards for the publication of Web Services enable the invocation of services having simple interaction protocols, but they fail to support complex e-business interactions, where the peers exchange several messages. In order to extend the classes of services which can be invoked by the consumers, we propose a conversational model supporting the management of complex interactions between clients and Web Services. Our model supports the consumer in the management of a conversation which respects the business logic of the service without imposing the explicit management of the conversational context.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability—*Interface definition languages*

General Terms

Languages

Keywords

Conversational agents, Web Services

1. INTRODUCTION

Web Services are subject to several limitations that reduce their applicability to realistic domains. For instance, the management of the interaction between the client and the service provider is difficult, unless very simple tasks are requested. In fact, the emerging communication standards, such as WSDL [23], support simple interactions, structured as question-answer pairs, but they are not expressive enough to define conversations including more than two turns. The fruition of services may require non trivial conversations between the consumer and the provider. For instance, during the interaction with a Web Service selling medium complexity, configurable items (e.g., bicycles), the specification of the item features may require more than one interaction step; moreover, failures may occur and have to be repaired before the solution for the consumer is generated. Finally, the Web Service might need to suspend the interaction, e.g., waiting for a sub-supplier, or a human operator to contribute to the generation of the solution.

This paper presents a conversational model aimed at extending the conversational capabilities of Web Services, by supporting complex interactions, where several messages have to be exchanged before the service is completed, and the conversation may evolve in different ways, depending on the state and the needs of the two participants. The proposed model has been defined by taking the speech-act theoretical model of dialog management [17, 7] as a starting point. However, it has been simplified to take into account two main factors: on the one hand, the types of interaction to be supported are complex, but not as flexible as human-computer dialog; on the other hand, the open Web environment imposes important scalability requirements.

We assume that the matching phase between service description and request has been performed and we focus on the service execution phase, by providing an explicit representation of the possible flow of the interaction. Our approach separates the specification of the conversational model from the implementation of the peers (e.g., the internal workflow specification), in order to support a simple specification of their external behavior, which can be suitably bound to the peers' internal implementation; see also [22] and [12]. Each peer should conform to the specified conversational behavior, depending on the role it fills (client or supplier). In particular, we propose a framework that enables the provider to assist the consumer during the service invocation in order to satisfy two main goals: first, we want to guarantee that the consumer respects the constraints imposed by the business logic declared in the provider's conversational flow. Second, we want to make this task as light as possible for the consumer, in order to enhance the applicability of our approach.

Section 2 provides background on the Web Services communication. Section 3 presents a conceptual dialog model based on speech acts. Section 4 proposes a framework for the lightweight management of conversations with Web Services, specifying the representation language and a framework for the server-side management of the interactions. Section 5 discusses some related work and closes the paper.

2. BACKGROUND

In order to improve the description of services in the Web,

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

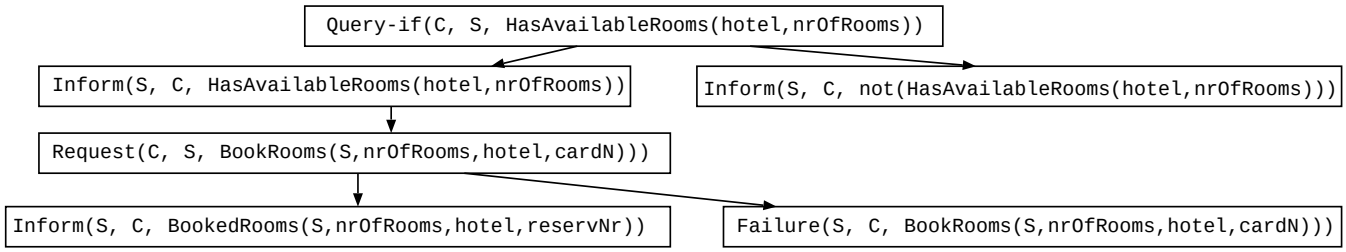

Figure 1: Speech-act based interaction flow for the hotel booking service.

the Semantic Web community is defining standards for the representation of semantic Web Services [15, 18]. These standards propose semantic representations of the interface and structure of a service and might represent the future approach to the publication of services in the Web. However, at the current stage, the semantic approach is still under development and a simpler approach, based on the current specification standards, is needed to improve the interaction with Web Services. In this perspective, an important issue is the support for a flexible type of communication between service consumers and providers. This issue has to be addressed to provide non-trivial services and can be handled by offering an effective mechanism to help the consumer to interact with the provider in the correct way, i.e., by respecting the constraints imposed by the business logic of the service.

The emerging standards for the definition of Web Services, such as WSDL [23], support the publication of elementary services. However, when complex services are defined, the flow of the operations to be performed cannot be specified. Therefore, the client may send the requests in the wrong order, possibly violating service preconditions which might cause the failure of the overall service. To regulate the peer-to-peer interaction, an explicit specification of the possible sequences of messages to be exchanged is needed. In the multi-agent systems research, this issue has been addressed by defining coordination protocols such as the FIPA Contract Net [10], and JAFMAS conversations [6]. However, the application of these approaches is not straightforward because it requires the agreement on specific communication languages and interaction protocols, such as in the AgentCities project [1]. Within the multi-agent systems area, the definition of middle agents has also been proposed to provide intelligent mediation services and coordination capabilities [14]. However, middle agents mainly focus on the flexible semantic matching between service descriptions and requests, leaving the possibly complex execution of the selected service as an open issue.

In the following subsections, we describe our proposal by introducing two alternative approaches to the representation of the conversation flow. Moreover, we sketch a framework for the development of conversational Web Services which we are implementing.

3. A SPEECH-ACT BASED REPRESENTATION OF CONVERSATIONS

Traditionally, social behavior of human and software agents in task-oriented interactions has been described by exploiting finite state automata [19]; moreover, hierarchical scripts and plan-based approaches have been introduced to model more complex, goal-oriented behavior; e.g., [5, 16, 10]. If we consider Web Services and their clients as interacting agents, the same approaches could be considered for modeling their conversational behavior. As the action-based conversational management approach is particularly clear, we will first use it to describe our proposal at a conceptual level, even though it will not be applied as it is for the reasons explained at the end of this section.

As a concrete example for the specification of conversation flow, we consider an hotel booking service: in order to book a room, the client is expected to start the interaction by asking if there are any available rooms. If all the rooms are booked, the interaction must stop. Otherwise, the client can provide the possibly private information needed to complete the transaction, such as the credit card number. Figure 1 shows a possible representation of the admissible turn sequences in the hotel booking service. Each action specifies a speech act representing a conversational turn performed by one of the peers; we have named the speech acts according to the FIPA specifications. We consider two main conversational roles: the client (consumer, C) and the service provider (S). The first parameter of a speech act represents the role filler that should perform the speech act, i.e., the agent sending the message. The second parameter denotes the recipient and the third parameter represents the content of the speech act. The arcs represent the possible turn sequences and the actions having more than one output arc represent situations where alternative speech acts can follow a certain turn, i.e., the responding peer is expected to perform only one of the turns. The interaction starts with the Query-if action, where the consumer (C) asks the service provider (S) whether the hotel has at least *nrOfRooms* available rooms (HasAvailableRooms(hotel, nrOfRooms)). The service provider may answer positively (Inform(S, C, HasAvailableRooms(hotel, nrOfRooms))), or negatively (Inform(S, C, not(HasAvailableRooms(hotel, nrOfRooms)))). In the second case, the interaction has to be terminated, as there is no way to book the rooms. In the first case, the client may request to book a certain number of rooms, by providing a credit card number (Request(C, S, BookRooms(S, nrOfRooms, hotel, cardN))). At this point, there are two possible continuations: the service provider notifies the client that the rooms have been booked and provides a reservation number (Inform(S, C, BookedRooms(S, nrOfRooms, hotel, reservNr))), or the booking action has failed (Failure(S, C, BookRooms(S, nrOfRooms, hotel, cardN))). The failure is a possible continuation because, although the room availability was checked before starting the transaction, in the meanwhile, some rooms might have been booked by other customers. Also exceptions might be handled: for

instance, the customer might enter a wrong card number, or the card might be blocked.

The interaction flow specified in Figure 1 is simpler than the typical (hierarchical) plans used in human-computer task-oriented dialogs (see [5]), because we need to model much simpler and predictable types of interaction. In particular, we have not specified any conditions on the actions of the pre-compiled action sequences in order to minimize the amount of information that has to be understood by the consumer. A peer may choose a specific course of action on the basis of its internal state, which may depend on the (un)successful execution of actions that the other peer does not need to be informed about. For instance, if the service provider informs the client that there are (not) available rooms, the client does not need to know the reason determining the answer: the only relevant fact is the performed speech act. In this way, the only type of information to be understood by both partners is the correct sequence of speech acts that can be executed.

The described conversational model enables a consumer agent to require the services provided by another agent by conforming to a specific interaction flow. In fact, if we assume that the service provider publishes the conversation script, the client may interpret it and start the interaction by performing the first turn specified in the script. Then, depending on the provider's reaction, the client can perform the correct turns, until the end of the interaction. However, this approach is not applicable to the current situation of Web Services, for the following reasons:

- The imposition of speech acts on Web Services, now publishing services by means of very simple languages (RPC invocations or WSDL services) is not realistic;
- The client is required to interpret both the speech acts and the conversation script published by the service provider. Moreover, to identify the admissible reactions to the provider's turns, the client has to maintain an internal representation of the interaction context: for example, the current focus of the conversation, i.e. the last executed action. The last requirement imposes supplementary efforts on the consumer than those required to invoke standard Web Services.

To support lightweight conversations, we propose a different approach, which uses emerging standards for the representation of the workflow and relies on the service provider for the control of the interaction.

4. LIGHTWEIGHT CONVERSATION MANAGEMENT

We propose that the service provider guides the consumers in the fruition of services by specifying, at each step of the interaction, the set of eligible turns they may perform. In this way, each client can rely on explicit instructions to correctly invoke the providers. Of course, to provide a picture of the admissible interaction flows, the service provider could publish both the service and the flow descriptions. This information represents documentation and can be used by the client to suitably bind the service invocations to its own business logic.

Each service has to be invoked with an initial request, performed by the client to trigger the interaction with the service provider. When the provider receives an init request, it responds by performing an agreement notification (or a rejection message if the interaction cannot be carried out).[1] In the same message, the service provider also specifies the set of alternative turns that the client may reply with. Such turns are invocations of the operations that the service provider can perform at that stage of the interaction. For instance, as far as the hotel booking service is concerned, after the agreement notification, the client may invoke the room availability check. After a confirmation that there are enough available rooms, the client can book the rooms, and so forth.

When the client receives the response, it has to retrieve the eligible operations, specified in the message. Then, the client has to choose the one to be invoked, sending back the request. The conversation continues in the same way, with the service provider performing the requested action and sending back results and possible continuations.

From the viewpoint of the client, this approach supports a seamless management of the interaction with the service provider. In fact, it only imposes the init operation as the initiation of any conversation, and the retrieval, from each incoming message, of the set of eligible continuations to choose from. The eligible operations are specified in the individual messages because they change along time. In contrast, the definition of the services needed to invoke them is provided in the public service specification.

The service provider, in addition to managing its own activities, has to guide the conversation with the client by computing the eligible reactions to its own turns. To this extent, the service provider has to maintain an explicit interaction context, storing the interaction status, and the last performed turn.[2] However, this is not an extra-overhead for the service provider, which has anyway to know the status of the interaction, in order to carry out the activities related to the requested services.

4.1 Simplified specification of conversations

To summarize, we propose that:

- The service provider publishes the services by specifying the operations that can be invoked by the clients;
- The service provider maintains a local interaction context, for each active conversation.
- Within each individual conversation, the service provider informs the client about the admissible turns it may perform, at each step.

The services can be published in a standard format, such as RPC interfaces, or WSDL operations. Moreover, the messages to be exchanged by the peers can be seamlessly extended with turn management information. Being implemented as SOAP messages, such information can be added as new parts in the message body [21]. Thus, only the representation of the conversation flow has to be simplified, with respect to the speech-act based approach, in order to make

[1]This extends the flow in Figure 1: the init message is the new root node and has an output arc entering the service provider's agreement message. The agreement message has an output arc entering the room availability check node.

[2]As the service provider may communicate with multiple clients in parallel, a separate interaction context has to be maintained for each active conversation.

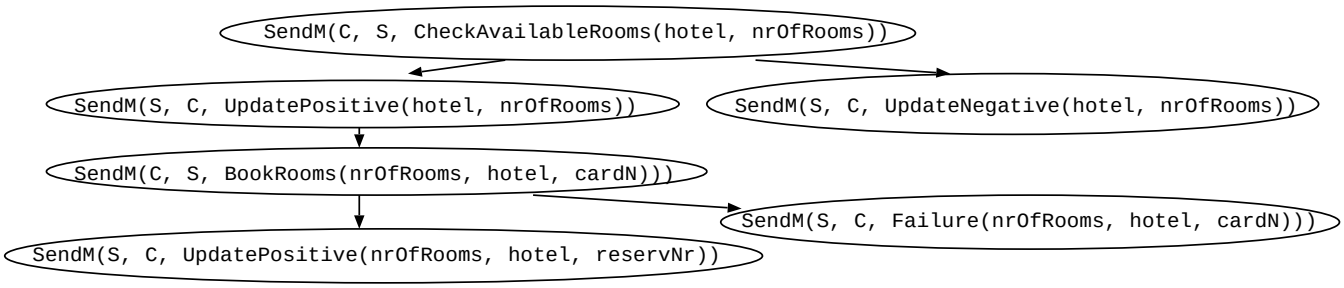

Figure 2: Portion of the simplified workflow description for the hotel booking service.

our proposal applicable to real cases. The specification of the interaction flow can be simplified as follows:

- The interaction turns can be modeled as generic conversational activities, by omitting the performed speech act (Inform, Query-if, etc.). Each turn corresponds to a send message (sendM) activity, whose parameters specify the actor (the sender of the message), the recipient and the content (involved operation). For instance, Query-if(C, S, HasAvailableRooms(hotel, nrOfRooms))) in Figure 1 is replaced with SendM(C, S, CheckAvailableRooms(S, hotel, nrOfRooms)), where CheckAvailableRooms is a domain-specific operation. Moreover, the Inform(S, C, HasAvailableRoom(hotel, nrOfRooms)) is replaced by SendM(S, C, UpdatePositive(hotel, nrOfRooms)), where UpdatePositive (UpdateNegative) is a belief revision operation: the beliefs of the peer performing the operation are changed as specified by its arguments.

- The actors of the operations specified as the third argument of the sendM activities can be omitted, because they coincide with the recipients of the messages.

Figure 2 shows the simplified representation of the interaction flow. Although this representation does not support the specification of different types of speech acts, with their semantics [20], the send message activities can be seen as requests to perform domain-specific operations; thus, the interaction is modeled as a sequence of turns where one of the peers requires that the other peer performs an operation. The conversational activity is clearly separated from the domain-specific behavior: similar to the speech-act based approaches, a send message activity specifies the required domain-specific behavior at the object level, as one of its arguments [7]. Thus, the conversational turns specified in the diagram of Figure 2 can be understood by the consumer, provided that the service provider describes in some way (even as natural language annotations) the meaning of the object level operations. Notice also that each turn is an asynchronous message, which one of the peers performs to carry the conversation one step forward. If the partner that should perform the next turn does not respond, the interaction is suspended, with time out.

4.2 WSFL-like specification of a conversation

We decided to represent the conversation flow by exploiting, as far as possible, an emerging standard for the representation of workflow in Web Services: the IBM's WSFL

```
<types>
<schema targetNameSpace=
       "http://example.com/hotelBook.xsd"
 lns="http://www.w3c.org/2000/10/XMLSchema">

  <element name="Book">
    <complexType>
      <all>
        <element name="hotel" type="string"/>
        <element name="nrOfRooms" type="integer"/>
        <element name="cardN" type="integer"/>
      </all>
    </complexType>
  </element>

  <element name="PositiveResult">
    <complexType>
      <all>
        <element name="res" type="string"/>
      </all>
    </complexType>
  </element>

  <element name="Exception">
    <complexType>
      <all>
        <element name="res" type="string"/>
        <element name="comment" type="string"/>
      </all>
    </complexType>
  </element>
</schema>
</types>
```

Figure 3: XML-schema definition of data types.

[13] service composition language.[3] This language enables a Web Service consumer to integrate the services offered by multiple providers by specifying several types of information. For instance, it includes an explicit declaration of the roles to be filled in the workflow and of the binding of such roles to concrete Web Service providers. Moreover, it supports the definition of state diagrams specifying the partial order relations between the services (operations) to be composed. Finally, it relies on WSDL for the definition of the integrated service and of the bindings to specific communication protocols, such as SOAP over HTTP. As WSFL is aimed at service composition, its syntax has to be extended

[3]WSFL has evolved in the Business Process Execution Language for Web Services (BPEL4WS)[8], which merges WSFL and Microsoft XLang and is positioned to become the Web Services standard for composition.

```
<message name="BookRooms">
  <part name="body" element="xsd:Book"/>
</message>

<message name="OK">
  <part name="body"
        element="xsd:PositiveResult"/>
</message>

<message name="Fault">
  <part name="body" element="xsd:Exception"/>
</message>

<portType name="HotelBookingPortType">
  <operation name="Booking">
    <output message="tns:BookRooms"/>
  </operation>

  <operation name="Confirmation">
    <output message="tns:OK"/>
  </operation>

  <operation name="Error">
    <output message="tns:Fault"/>
  </operation>
</portType>
```

Figure 4: WSDL representation of operations.

to represent conversation-specific activities. For instance, WSFL only defines supplier roles, because it corresponds to the viewpoint of the service integrator. However, in a conversation, both the client and the service provider are active partners and must be represented. In spite of the required syntax extensions, we decided to use such a language, as it represents an already defined proposal, instead of introducing a totally different representation language.

In the following, we describe a WSFL-like representation of the hotel booking service. We first describe the WSDL operations; then, we describe the conversation flow by composing such operations. We assume that the client is able to interpret a WSDL specification, while it does not need to know WSFL. The WSFL specification is used by the service provider to manage the interaction flow by employing a suitable workflow engine.

4.2.1 WSDL specification of operations

Figures 3 and 4 show the WSDL representation of the hotel booking service. More specifically, Figure 3 shows a portion of the XML-schema defining the structure (action-name and arguments) of the object level actions. For instance, the Book action has three arguments: the *hotel* (a string), the *nrOfRooms* and the *cardN* (two integer variable). Moreover, the PositiveResult action has one argument *res* storing the description of the result of an event. Such actions are the signatures of methods to be executed by a peer when it receives a message from its conversational partner. For instance, the Book action corresponds to a method used by the service provider to reserve a number of hotel rooms. Moreover, the PositiveResult and NegativeResult actions are belief revision methods aimed at collecting positive or negative results.

Figure 4 shows the abstract definition of some operations published by the service; moreover, it shows the port type grouping the operations. Each operation is characterized by a name and a message to be handled. The message is specified as follows:

- The *structure* of the message is defined by specifying the type of the action to be included in its body. For instance, the message to be handled in the Booking operation has as its body an action of type BookRooms.
- The *direction* of the message is specified: in the standard WSDL specification, all the operations are described from the service provider's viewpoint and they include one or more messages between the peers. In particular, there are *input* and *output messages*, where input messages represent the messages to be received by the service provider and output messages represent those that the provider sends to the client. For instance, a typical WSDL operation includes an input message requesting the execution of a method and an output message returning the result to the client.
  In our approach, we avoid assembling multiple turns within the same operation. Moreover, the operations can be performed either by the service provider, or by the client. Thus, all the messages are declared as output messages as they are, indeed, messages which the actor sends to the recipient.

4.2.2 WSFL-like specification of conversation flow

Given the WSDL specification of the operations, the interaction flow can be represented by defining the conversational activities to be performed by the peers and the partial order relations between such activities. The activities correspond to the send message actions (sendM) shown in Figure 2. Each activity is characterized by a name and the arguments: the actor, the recipient and the content operation, which represents the object level operation requested by sending the message to the peer. Figure 5 shows a portion of the flow model for the hotel booking service:[4]

- The "serviceProvider" and "client" specifications characterize the two conversational roles and the peers filling such roles.
- Some of the activities to be carried out by the peers are defined by specifying the requested WSDL operation and the actor that should perform the activity. For instance, activity "sendBookingRequest" has to be performed by the client and has as its content the "BookRooms" operation; the client requests that the service provider books the rooms.
- The controlLink specifications define the partial order relations between activities.

The operations associated to complex domain-specific actions, such as BookRooms, are triggered by the consumer by means of a message and are executed by the service provider. In contrast, the provider typically requests the client to perform confirmation, disconfirmation and error operations, needed to collect the results of the performed

[4]The "client" and "actor" labels are extensions to the original WSFL syntax. Moreover, no dataLinks are defined, because, different from the representation of a domain-level workflow, no sub-suppliers are invoked; thus, no instantiated parameters have to be passed in such invocations. Finally, some of the controlLink relations refer to activities which we could not specify due to the space constraints.

```
<flowModel name="HotelBookingFlow"
    serviceProviderType="HotelService">

  <serviceProvider name="hotelServer"
                   type="HotelServiceWS">
     <locator type="static"
              service="ourHotelServer.com"/>
  </serviceProvider>
  <client name="client" type=""/>

  <activity name="sendBookingRequest">
     <performedBy actor="client"/>
     <implement>
        <export>
           <target portType="hotelServicePT"
                   operation="Booking"/>
        </export>
     </implement>
  </activity>

  <activity name="sendBookingConfirmation">
     <performedBy actor="hotelServer"/>
     <implement>
        <export>
           <target portType="hotelServicePT"
                   operation="Confirmation"/>
        </export>
     </implement>
  </activity>

  <activity name="sendErrorMessage">
     <performedBy actor="hotelServer"/>
     <implement>
        <export>
           <target portType="hotelServicePT"
                   operation="Error"/>
        </export>
     </implement>
  </activity>

<controlLink source="sendAvailabilityRequest"
             target="sendConfirmation"/>
<controlLink source="sendConfirmation"
             target="sendBookingRequest"/>
<controlLink source="sendBookingRequest"
             target="sendBookingConfirmation"/>
</flowModel>
```

Figure 5: WSFL-like conversation flow specification.

tasks. This is the reason why, in the conversation flow, the activities performed by the client have domain-specific operations as their argument. In turn, most activities performed by the service providers are acknowledgments that the previously requested operations were (un)successfully performed, or results of such operations.

Figure 6 shows, at the conceptual level, the complete specification of the conversation flow of the hotel booking service. The nodes specify the send message activity to be performed in each turn. The sendM activity has the following arguments: the actor, the recipient and the name of the requested WSDL operation. Moreover, if the actor is the service provider, the sendM activity has a further argument, storing the list of the possible continuations of the conversation. As already mentioned, the turn-related information is stored in the body of the SOAP message, as a separate part. In the figure, we have added in bold the name of the requested WSDL operation, in order to help the reader to

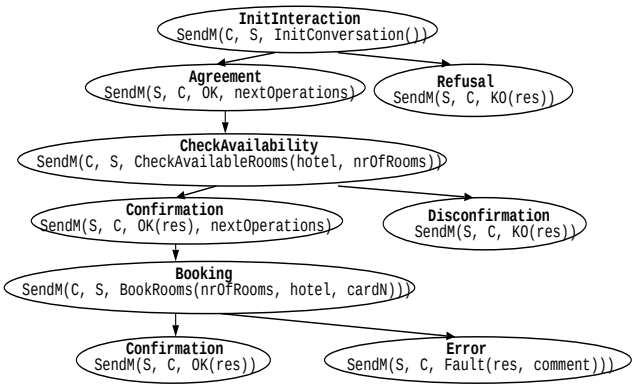

Figure 6: Complete conversation flow for the hotel booking service.

map the states to the related WSDL-WSFL specifications.

4.3 A possible framework for the server-side management of complex interactions

The current Web Services typically manage the interaction context by exploiting Java Servlets, or similar technologies in the Microsoft .net world, which maintain the status of the active client sessions within the service provider. Although the basic exploitation of Servlets as tools for the communication with a Web-based application is straightforward, the embodyment of the service's business logic is difficult, as it requires that the service exposes the operations to be invoked according to the correct conversation flow. Therefore, some three-tier architectures have been designed as multi-agent systems, where a flexible frontend is devoted to the management of the interaction with other agents. The frontend is itself complex and separates the dialog management activity, carried out by a conversational agent, from the support for Web communication and low-level session management (a Servlet); see [2, 3].

In order to manage complex conversations with the consumer applications, each Web Service provider should implement a module responsible for the management of the conversational flow, based on the respective WSFL description. This module represents the core of the Servlet listening to the clients' requests and invoking the appropriate components to execute the services (e.g., Enterprise Java Beans). The module would allow keeping track of asynchronous communication between the client application and the Web Service Provider[5]. A Dialog Flow state would be maintained for each client, to allow resuming the conversation when the possibly complex operation has been completed, sending back to the client the result and the next operation to perform. The proposed architecture of the Web Service Provider would be similar to the one shown in Figure 7: a Servlet would support the (SOAP) HTTP-based communication with the client, by catching the incoming requests and forwarding them to an Interaction Flow Manager for their management. Different types of components, an internal one like Component A in the Figure, or an external Supplier, can be invoked to provide the service. The

[5]Currently, handling the losely coupled (asynchronous) Web Services is declared to be one of the biggest challenges for the Web Services infrastructures industry.

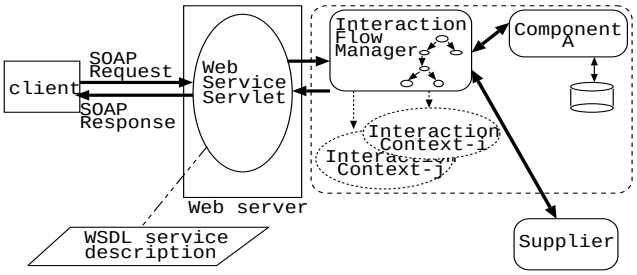

Figure 7: Interaction with a web service provider.

Servlet would also send back the SOAP response messages to the client application. The Interaction Flow Manager would exploit an internal representation of the possible interaction flows in order to compute the possible continuations of each interaction. The figure shows a situation where the Interaction Flow Manager is managing two parallel interactions, and thus maintains two interaction contexts: i and j. The Interaction Flow Manager invokes an internal component, e.g., an EJB, or an external supplier to execute the possibly complex operations necessary to complete the service.

The approach to the representation of the conversation flow we described in the previous sections supports the unambiguous specification of the interaction flow between the client and the Web Service provider. Therefore, it can be used as the basis for the development of a tool that leverages the implementation of a conversational agent taking care of the server-side management of interactions with the clients. In particular, a Web Service Provider Framework can be designed to make the implementation of the Interaction Flow Manager straightforward, given the sequence of the conversational turns. This framework should get in input the WSFL specification of the interaction flow and generate the skeleton of the module that handles such a flow. We will use the IBM's Web Services Description Language for Java Toolkit (WSDL4J), which allows the creation, representation, and manipulation of WSDL documents describing services [11, 9], to process the WSDL part of the description. Therefore the module will navigate the interaction flow representing the conversation steps.

4.4 Application to typical WSDL service descriptions

The proposed conversational model could be applied to current Web Services with limited effort, if they already offer a WSDL service specification. In particular, the service descriptions could be extended to specify the conversation flow, at a limited level of detail, by adding the WSFL-like flow specification. Figure 8 shows a possible representation of the hotel booking service, based on the typical WSDL representation style, where the operations include multiple input, output and fault messages. In particular, the CheckAvailability2 operation describes the set of messages to be exchanged between the peers to check the room availability: an input message specifying the CheckAvailableRooms action; an output message confirming the availability of the rooms (OK action); a fault message (KO action). Operation Booking2 includes other three similar messages. The service specification is based on the same data types defined in Figure 3.

Although the definition of the operations is very different

```
<portType name="HotelBookingPortType2">
  <operation name="CheckAvailability2>
    <input message="tns:CheckAvailableRooms"/>
    <output message="tns:OK"/>
    <fault message="tns:KO"/>
  </operation>
  <operation name="Booking2>
    <input message="tns:BookRooms"/>
    <output message="tns:OK"/>
    <fault message="tns:KO"/>
  </operation>
</portType>
```

Figure 8: Standard WSDL representation of the hotel booking service.

from the one we proposed in Section 4.2, the WSFL-like representation could be used to specify the correct sequence of operations to be invoked also in this case. In particular, to support the client in the selection of the first operation to be performed, an initConversation WSDL operation should be defined, which includes an input (init) message, an output (OK) message, and a fault (KO) message. Moreover, the WSFL activities should be defined, by mapping an activity to each WSDL operation. For instance, a checkAvailRequest activity could be mapped to the CheckAvailability2 operation; moreover, a bookRequest activity could be mapped to the Booking2 operation, and so forth. Finally, the partial order relations between the whole set of operations should be defined: the initConversation activity must be performed before checkAvailRequest, which in turn must be performed before bookRequest. Given this specification, the service provider could exploit the proposed framework to update the management of the interaction flow and guide the consumers in the correct service invocation.

5. DISCUSSION

The specification of the conversation flow is central for the Web Services research area. As pointed out in the previous sections, when a service provider publishes more than one operation, the client has to guess the correct sequence to be followed in the invocation of such operations. However, the coordination between client and service provider may be a problem even during the execution of a single operation, if it includes more than one message, as *request-response* and *solicit-response* operations do. More specifically, if more than one (input, output or fault) message is present, the correct interaction pattern cannot be determined from the service specification itself. For instance, consider the CheckAvailability2 operation in Figure 8. The three messages included in the operation form a subset of the conversation flow: the first message should be performed before the other two; moreover, the second and the third one are alternative messages, that the server provider sends to the client, depending on the result of the CheckAvailableRooms action. Our proposal supports the specification of the interaction flow at the granularity level of the individual messages (conversational turns). Thus, it solves the coordination problem both at the level of the invocation of operations, and within the operations themselves.

Other XML-based standards for the specification of the conversation flow in e-business interactions with Web Services have been recently proposed and are currently sub-

mitted as W3C standards. For instance, WSCL (Web Services Conversation Language [22]) and WSCI (Web Services Choreography Interface [4]) introduce an explicit representation of Web Services interaction processes, aimed at defining the admissible sequences of messages to be exchanged by the peers. To this purpose, WSCL exploits a sequence diagram model, which the peers should interpret to handle the conversation, while WSCI introduces the notion of interaction process, with the specification of timing constraints on the service invocation. Moreover, cpXML (IBM's Conversation Support [12]) introduces an explicit notion of Conversational Policy, as a a machine readable specification of a pattern of message exchange in a conversation, which can be used to make the interaction with complex Web Services easier from the client application viewpoint.
Our model differs from the above mentioned ones because they assume that each peer separately maintains its own internal record of the conversation state, while we propose that only the service provider maintains the state of the conversation and suggests the following step to the client application. Moreover, our approach differs from both the WSCL and WSCI ones in two aspects: first, they conform to WSDL in the specification of *request-response*, *solicit-response*, etc., operations; thus, they do not support a fine-grained specification of the conversational turns. Second, they specify the conversations from the viewpoint of the service provider. Thus the client has to interpret the specifications in the reverse perspective. For example, an input message for the provider is an output one for the client.

Our proposal builds on the conceptual model of speech acts [7] for the specification of the dialog between the peers. This model represents the peers' dialog acts as actions performed by an actor towards a recipient. Moreover, the model clearly separates the conversational activity from the execution of the object level actions the partners "talk about" during the interaction. The explicit representation of the peers and of their conversational activity (send message activities) supports a detailed and unambiguous specification of the possible sequences of interaction turns, at the granularity level of the individual messages to be exchanged.

The proposed WSFL-like specification of the conversation flow supports the specification of services requiring complex interactions between clients and suppliers. The hotel booking example is a relatively simple case, selected to simplify the presentation of the overall approach.

We thank Marino Segnan for his contributions to our work. This research has been partially funded by Consiglio Nazionale delle Ricerche within project CNRG0015C3.

6. REFERENCES

[1] Agentcities. Agentcities network services. http://www.agentcities.net/, 2002.

[2] L. Ardissono, C. Barbero, A. Goy, and G. Petrone. An agent architecture for personalized Web stores. In *Proc. 3rd Int. Conf. on Autonomous Agents (Agents '99)*, 182–189, Seattle, WA, 1999.

[3] L. Ardissono, A. Goy, G. Petrone, and M. Segnan. A software architecture for dynamically generated adaptive Web stores. In *Proc. 17th IJCAI*, 1109–1114, Seattle, WA, 2001.

[4] A. Arkin, S. Askary, et al. Web Service Choreography Interface 1.0. http://ifr.sap.com/wsci/specification/wsci-specp10.html.

[5] S. Carberry, J. Chu-Carroll, and S. Elzer. Constructing and utilizing a model of user preferences in collaborative consultation dialogues. *Computational Intelligence*, 15(3):185–217, 1999.

[6] D. Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, University of Cincinnati, Stanford, CA, 1997.

[7] P.R. Cohen and H.J. Levesque. Rational interaction as the basis for communication. In P.R. Cohen, J. Morgan, and M.E. Pollack, eds., *Intentions in communication*, 221–255. MIT Press, 1990.

[8] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, v. 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[9] IBM developerWorks. Web Services Description Language for Java Toolkit (WSDL4J). http://www-124.ibm.com/developerworks/projects/wsdl4j/.

[10] FIPA. Foundation for Physical Intelligent Agents. http://www.fipa.org/.

[11] Object Management Group. CORBA 2.4.2 specification. http://www.omg.org.

[12] J.E. Hanson, P. Nandi, and D. Levine. Conversation-enabled web services for agents and e-Business. In *Proc. of the Int. Conf. on Internet Computing (IC-02)*, 791–796, Las Vegas, 2002.

[13] IBM. Web Services Flow Language. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[14] M. Klusch and K. Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, eds., *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 8, 197–224. Springer-Verlag, 2001.

[15] S. McIlraith, T.C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[16] C. Rich, D. McDonald, N. Lesh, and C. Sidner. Collagen: Java middleware for collaborative agents.

[17] J.R. Searle. Indirect speech acts. In P. Cole and J. Morgan, eds., *Syntax and Semantics: Speech Acts*, vol. 3, 59–82. Academic Press, New York, 1975.

[18] DAML Services Coalition. DAML-S: Web Service description for the Semantic Web. In *Int. Semantic Web Conference*, Chia Laguna, Italy, 2002.

[19] A. Stein and E. Maier. Structuring collaborative information-seeking dialogues. *Knowledge-Based Systems*, 8(2-3):82–93, 1994.

[20] D. Steiner. An overview of FIPA 97. http://www.cselt.it/fipa/#Papers, 1997.

[21] W3C. Simple Object Access Protocol (SOAP) v. 1.2. http://www.w3.org/TR/2001/WD-soap12-20010709/.

[22] W3C. Web Services Conversation Language (WSCL). http://www.w3.org/TR/wscl10, 2002.

[23] W3C. Web Services Definition Language. http://www.w3.org/TR/wsdl, 2002.