A CONVERSATIONAL APPROACH TO THE INTERACTION WITH WEB SERVICES

LILIANA ARDISSONO, GIOVANNA PETRONE AND MARINO SEGNAN
Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149 Torino, Italy
Phone: +1 011 6706711
Fax: +1 011 751603

Abstract

The emerging standards for the specification of Web Services support the publication of the static interfaces of the operations they may execute. However, little attention is paid to the management of long-lasting interactions between the service providers and their consumers. Although this is not an issue in the case of "one-shot" services, it challenges the provision of services requiring the exchange of multiple messages between the business partners.

In this paper, we present a conversation model supporting the management of long-lasting interactions where several messages have to be exchanged before the service is completed. Our model aims at facilitating the consumers during the service invocation because in this way the establishment of short-term business relations can be simplified. To this extent, we provide a computational framework that can be exploited to manage a conversation between the consumer and the service provider. Our framework is inspired from the research developed in Computational Linguistics and in the area of Multi-Agent Systems to manage human-to-computer and agent-to-agent dialog. However, we employ techniques suitable to comply with the emerging Web Service standards and with the scalability requirements of the Internet.

keywords: Web Services, Web agents for producers and consumers

1 INTRODUCTION

As described in [Curbera et al., 2002a], *"Web Services are emerging to provide a systematic and extensible framework for application-to-application interaction, built on top of existing Web protocols and based on open XML standards."* Although Web Services are aimed at providing standard interfaces for the interoperation of legacy software in the Internet, they are currently subject to several limitations that reduce their applicability to realistic cases. For instance, the emerging service publication standards, such as WSDL [W3C, 2002b], support the specification of the static interfaces of elementary services. However, the management of the interaction between the consumer and the service provider is difficult, unless simple services are requested, because these standards do not enable the service provider to specify the order of the operations to be invoked by the consumer. Moreover, these standards only 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, such as those related to the customization of complex products, where the list of features to be configured has to be decided at run time. As a matter of fact, the development of Web Services with rich interaction capabilities is interesting because one could aim at:

- Making a personalized service, such as a recommender system, available as a Web Service. For instance, a movie recommender could be extended with a public interface that enables digital personal assistants to invoke it on behalf of their users. Moreover, a bank might offer a loan customization service that can be exploited by distributed commercial applications to negotiate financial support for their customers.

- Composing Web Services in a consumer application serving human users. For instance, middle agents, such as real estate agents and car sellers, could develop Web-based applications supporting a completely automated interaction with the customer, from the selection of the good to be purchased to the contract definition; e.g., see [McIlraith et al., 2001]. Similar to the traditional scenario, populated by human operators, the middle agent would manage a complex workflow, invoking and orchestrating different services, such as insurance agencies, attorneys, banks and/or financial agencies.

The current work on workflow management is focused on the service composition in the Web but the proposed approaches assume a very simple type of interaction with the suppliers. For instance, BPEL4WS [Curbera et al., 2002b, Curbera et al., 2003] supports the specification of complex service compositions. However, the management of the interaction between the provider and the consumer mainly deals with low level communication issues such as transaction management; see [Cabrera et al., 2002]. In contrast, in order to make the service fruition possible even when the involved suppliers require complex interactions, the invocation has to be modeled as a conversation where the participants may exchange several messages before the service is completed; e.g., requirements acquisition, negotiation and other types of interaction. For instance, during the interaction with a Web Service supporting the configuration of medium complexity products, the specification of the item features may require more than one step; moreover, failures can occur and have to be repaired before the solution for the consumer can be generated. Finally, in some cases, the Web Service may require to suspend the interaction, e.g., waiting for a sub-supplier or a human operator to contribute to the generation of the solution. Similarly, the consumer might suspend the interaction because it needs supplementary

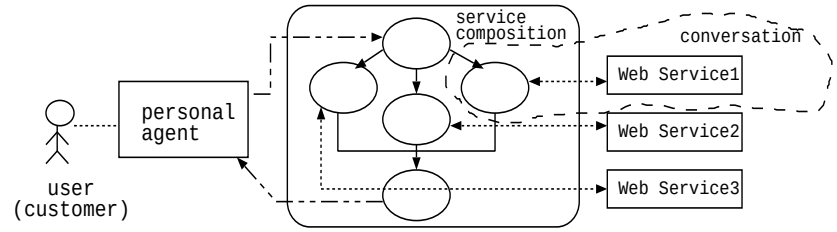

Figure 1: Service composition and conversation.

information from the customer before choosing certain product features.

A loosely coupled approach to the management of the interaction is needed to support successful business interactions and, in the meantime, enable the consumers to suitably 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, at least at the consumer side. In fact, the consumer may need to start several e-business interactions with heterogeneous providers. Therefore, it should not be required to manage tightly coupled interactions with each of them, especially when they are held outside a well established B2B relationship.

In this paper, we present a conversation model for Web Services, aimed at 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 states of the two participants. We have defined our model by taking the speech-act theoretical model of dialog management as a starting point [Searle, 1975, Cohen and Levesque, 1990]. However, we have simplified it to take the emerging Web Service publication standards into account and to develop an effective conversation management framework. The rest of this paper is organized as follows: Section 2 describes the assumptions underlying our proposal and positions it in the Web Services research. Section 3 describes a speech-act based representation of conversation flow and Section 4 presents our conversation model. Section 5 sketches our infrastructure for the development of conversational Web Service providers and consumers. Section 6 compares our proposal to the related work and Section 7 concludes the paper and outlines our future work.

2 PREMISES

A flexible conversation model is needed to support the dynamic invocation of Web Services and the present paper contributes to the definition of this model. Before presenting our proposal, we discuss three aspects required to understand our contribution.

First of all, 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 noticed 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 [Kamamura et al., 2003]. 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. The binding phase could lead the consumer to conform to a portion of the domain ontology exploited by the Web Service.

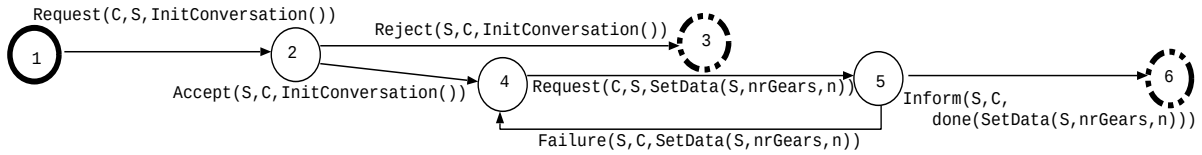

Figure 2: Speech-act based interaction flow for a trivial product customization Web Service.

Second, the management of conversations should not be confused with the service composition handled by workflow engines, as these are complementary to one another; the conversation management enriches the flexibility in the invocation of the individual suppliers whose services are composed by the consumer. Figure 1 shows an application composing Web Services to provide a complex service to the user. The depicted graph shows the partial order relations between workflow states. Each state may include internal activities and invocations of external Web Services. The dotted lines represent the interaction between the consumer and the suppliers and may denote one-shot invocations of operations or complex conversations. The focus of our work is on the possibly complex and asynchronous conversation between a module of the application and the Web Service provider; see the highlighted area in the figure.

Third, our conversation model clearly separates the aspects concerning the internal implementation of a Web Service from the communication protocol defining the invocation of its operations. As noticed in [W3C, 2002a], no knowledge about the implementation of the providers should be needed to invoke them.

3 A SPEECH-ACT BASED APPROACH

The development of dialog models supporting the management of long-lasting communication derives from the Computational Linguistics research. Although the results achieved in this area were focused on the management of human-to-computer interaction, they provided a solid basis for recent communication models developed by the Multi-Agent research community.

Traditionally, social behavior of human and software agents has been modeled by exploiting speech-acts [Searle, 1975] that separate the illocutionary force of the agents' messages from the object-level actions underlying the execution of the conversation turns. In particular, Finite State Automata (FSA) have been applied in the management of task-oriented interaction, in order to specify the conversation roles to be filled in by the participants and the possible sequences of turns that may be performed; see [Stein and Maier, 1994]. Moreover, hierarchical scripts and plan-based approaches have been introduced to efficiently model goal-oriented behavior [Cohen and Levesque, 1990] and to separate the management of conversational behavior from the domain-dependent activities carried out by the agents [Rich et al., 2002]. These approaches assume that the agents play well defined conversation roles and they cooperate to a domain-level activity. Their social behavior is aimed at coordinating the internal processes; e.g., questions may be posed to verify the feasibility of the actions to be performed and notifications are made to notify their (un)successful execution. The speech-act based conversation models are suitable to describe communicative behavior at the conceptual level and they have been applied to the description of agent-to-agent communication as well; e.g., see KQML [Finin et al., 1995] and FIPA ACL [FIPA, 2000]. Thus, they could be applied to specify the conversation flow between Web Services and their consumers. In particular, the messages to be exchanged during the service fruition can be seen as conversational actions performed to carry out a task-oriented

dialog between entities filling the Service Provider and the Consumer roles.

As a concrete example for the specification of a conversation flow, we consider a trivial product customization problem, where the consumer interacts with a Web Service to set the number of gears of a bicycle. Figure 2 shows a FSA representing the admissible turn sequences in this service. The states of the automaton represent the dialog states: the plain circles denote the conversation states and the thick one (state 1) is the initial state. The thick dotted states (3, 6) are final dialog states. The speech acts are specified as labels of the arcs. Each speech act represents a conversation turn to be performed by one of the participants and is named according to the FIPA specifications. 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 states having more than one output arc represent mutually exclusive speech acts, i.e., the agent is expected to perform only one of them. The interaction starts with the *Request* turn, where the consumer C asks the service provider S to start the interaction (*initConversation*). The provider may accept to perform the request (*Accept(S, C, InitConversation())*), or reject it (*Reject(S, C, InitConversation())*). In the second case the interaction terminates. In the first one the consumer may request to set the number of gears (*Request(SetData(S, nrGears, n))*). If the Web Service successfully performs the operation, it acknowledges the consumer (*Inform(S, C, done(SetData(S, nrGears, n)))*) and the interaction terminates.[1] Otherwise, the Web Service notifies the failure of the operation and enables the consumer to set the feature again; see the loop between states 4 and 5.

The FSA specifying the conversation flow of a service provider could be exploited by the consumers to manage the turn-taking activity and to correctly invoke the operations on the provider. However, this approach is not desirable for two main reasons: first, the imposition of speech acts on Web Services, which now publish services by means of very simple languages such as RPC invocations or WSDL operations, is not realistic. Second, in order to select the admissible reactions to the provider's turns, the consumer has to maintain an internal representation of the interaction context that, at minimum, includes the active state of the automaton, i.e., the output state of the last speech act performed by the service provider. The second requirement is particularly problematic from the interoperability point of view because it imposes that the consumer locally executes a copy of the conversation automaton employed by the provider.

4 OUR PROPOSAL

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 invocation. More specifically, we propose that:

- The provider publishes the services by specifying the operations in a standard language. This is necessary to let the consumer bind the invocations of operations to its own business logic, e.g., by associating the invocations to its internal processes.

- The specification of the interaction flow is based on a flexible but simple representation formalism supporting the specification of the correct sequence of turns without the overhead of the pure speech-act model.

[1]The *done* operator was introduced by Cohen and Levesque to represent the state of the world after an action is successfully performed [Cohen and Levesque, 1990].

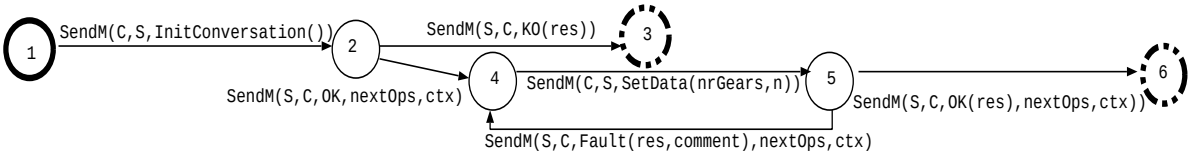

Figure 3: Simplified conversation flow specification.

- For each active conversation, the service provider maintains a local interaction context to guarantee that at least one of the participants controls the dialog.
- At each step, the provider enriches the messages it sends with contextual and turn management information in order to instruct the consumer about how to continue the conversation. The contextual information can be void in trivial interactions. The turn management information consists of the eligible turns (henceforth, next operations) that the consumer may perform to carry the interaction one step forward.

The first two points are aimed at guaranteeing that the representation formalism employed for the publication of services may be easily adopted by the suppliers. The other ones leverage the interaction management at the consumer side, therefore making the engagement in interactions with Web Services as seamless as possible.

4.1 SIMPLIFIED SPECIFICATION OF CONVERSATIONS

With respect to the approaches defined to manage agent-to-agent communication, we simplify the specification of the interaction flow by modeling the interaction turns as generic conversational activities where the performed speech act (*Request*, *Inform*, etc.) is omitted. This is possible because all the turns are requests that the sender performs to trigger the execution of the invoked operations on the receiver. Moreover, we extend the interaction turns with turn-management information needed to assist the consumer in the invocation of the operations offered by the service provider. Each conversational action represents a simplified speech act, where the sender asks the recipient to perform the operation specified as an argument. The conversational actions have the following arguments:

- The sender of the message, which may be the consumer C, or the service provider S.
- The recipient of the message (similar).
- An operation that the sender invokes on the recipient. Notice that 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 alternative operations offered by the provider which the consumer may invoke in the next conversation step.
- A context argument, storing information about the interaction state (*ctx*). Similar to the *nextOps* argument, *ctx* one is only present in the messages directed to the consumer.

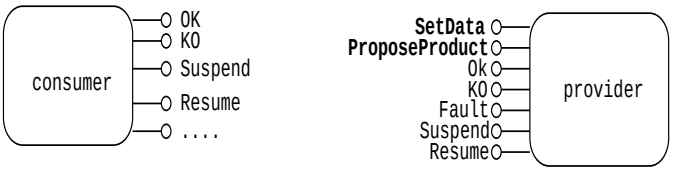

Figure 4: Domain-level operations and communicative actions.

Figure 3 shows the simplified representation of the interaction flow in our product customization service. The automaton has the same structure as the previous one, but the conversation turns are represented as send message activities (*SendM*). For instance, *Request(C, S, SetData(S, nrGears, n))* in Figure 2 corresponds to *SendM(C, S, SetData(nrGears, n)). Inform(S, C, done(SetData(S, nrGears, n)))* is replaced with *SendM(S, C, OK(res), nextOps, ctx)*. Moreover, the positive and negative acknowledgments are simplified to generic OK and KO actions. Two kinds of (object-level) operations may be the arguments of a conversation turn:

- *Domain-level operations*, such as $SetData$, representing domain-dependent operations to be invoked during the service execution.
- *Communicative actions*, such as $InitConversation$, OK, $Fault$, $Suspend$, $Resume$ and $ReceiveResult$, that are independent of the domain and are invoked during the service fruition to coordinate the behavior of the interactants.

Figure 4 shows the names of some object-level operations offered by a consumer application and by a product customization Web service. We have depicted the names of the domain-level operations (specific of product customization) in boldface to distinguish them from the communicative actions, which are applicable across different domains. Being concerned with the service execution, the domain-level operations are only offered by the service provider. In contrast, the communicative actions enable the interactions and thus they should be offered by the provider as well as by the consumer. For instance, the consumer must offer the *ReceiveResult* action, which corresponds to the WSDL *output messages* sent by the service provider to acknowledge the service execution and notify the results. Moreover, the $Suspend$ and $Resume$ actions are needed if the service execution has to be suspended to handle pauses and delays.

The proposed flow specification is not sufficient to characterize the conversation with highly interactive service providers. For instance, consider a Web Service customizing the configuration of products. The operations to be performed can be clearly defined, e.g., specifying the needed components and their features. However, the features whose values have to be set

```
SendM message:
Invoked action;
Next operations:
  Domain-level
            actions
  Communicative
            actions);


Context;
```

```
SendM message
Message: OK("")
Next operations:
 SetFeatures(nrOfGears);
 PostponeSet(nrOfGears);
 SuspendConversation(id);
 ....
Context:
  Current Focus:
         specify(gearBox)
  Achieved Activities:
         specify(frame)
         specify(handle)
```

Figure 5: Instructing the consumer about how to continue a conversation.

depend on the components required by the consumer. Therefore, the Web Service has to dynamically determine the correct invocation of operations during the exploration of the search space, by taking contextual information about the interaction into account. As a solution, we propose to relax the specification of the signatures of the operations by admitting generic arguments instantiated with the actual parameters during the service fruition. For instance, the *SetData(nrGears, n)* operation would be generalized to a *SetData(args)* one, with the $args$ argument bound to the actual parameters at service invocation time. Figure 5 sketches the format of the messages sent by the provider to the consumer during the customization of a bicycle.

- On the left side, the figure shows the abstract structure of the $SendM$ messages. The sender and receiver arguments are omitted because they are stored in the message header.
- On the right side, a sample message is shown. This is a positive acknowledgment (OK) generated by the provider to notify the successful execution of an operation. The provider also specifies that the consumer may invoke $SetData$ to set the number of gears, $PostponeSet$ to postpone the setting to a later stage of the interaction, or it may suspend the conversation. The *Current Focus* specifies that the execution of these operations is aimed at carrying out the specification of the gearbox. Moreover, the frame and the handle have already been specified (*Achieved Activities*).

The sample context object shown at the right side of Figure 5 (*Context*) sketches a possible representation of the fulfillment state of a service. During the interaction with a consumer, the context is enriched to show the progress in the product customization. We introduced the context argument to support the development of consumers displaying different levels of initiative during the interaction with the Web Service, while maintaining the management of contextual information at the service provider side. Although we cannot make any assumptions on the consumer's decision capabilities, the invocation of the operations can depend on contextual information about the previous part of the interaction. For instance, the consumer might condition the provision of the customer's credit card data to the fact that the product has been completely defined. The structure of the context argument depends on the application domain. For instance, the object can be empty in simple and deterministic interactions, where the next operation argument includes at most one element.

4.2 A USE CASE: CONFIGURATION OF PRODUCTS

In order to show the interaction requirements that can be satisfied by exploiting our conversation model, we focus on the configuration domain. The configuration use case is sufficiently general to cover different application scenarios, among which the customization of products and services, attracting a lot of attention in the CRM and e-commerce areas; e.g., see the research about mass customization [Piller and Schaller, 2002].

The configuration of a product may require the selection of components to be included in the product and the setting of the feature values. As the product features may be related by complex constraints, this activity is usually carried out by employing a configuration system; e.g., see [Mailharro, 1998, ILOG, 2002]. In our work, we assume that the service provider, i.e. one of the actors, has such a system and runs it during the interaction with the consumer (the other actor) in order to elicit the information needed for the configuration process, step by step, and return the results. The specification of the conversation flow is thus aimed at guaranteeing that

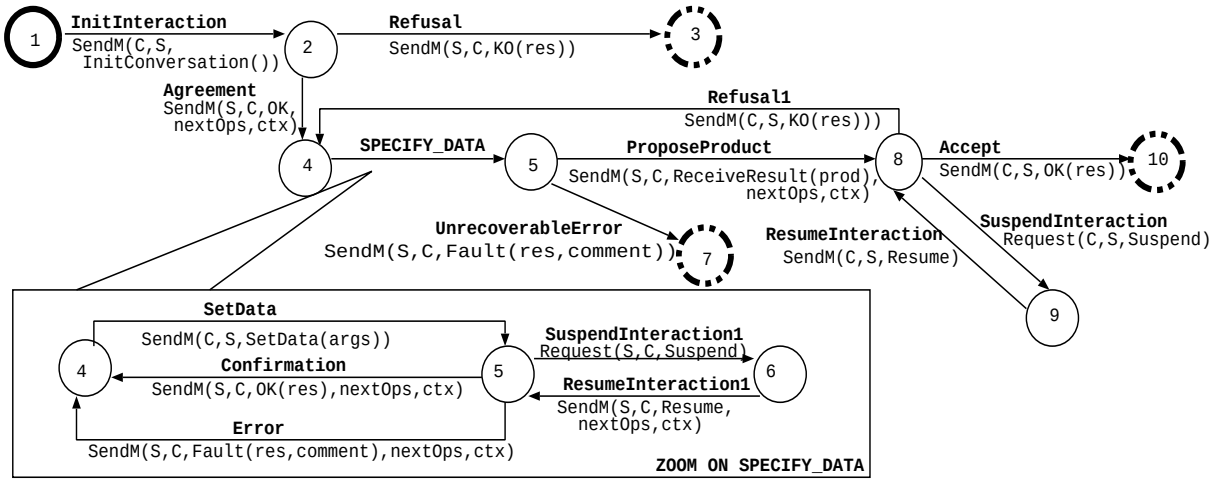

Figure 6: Conversation flow specification of a product customization Web Service. The portion of automaton in the square is a zoom on the SPECIFY_DATA arc.

the provider elicits the needed information from the consumer and that the consumer negotiates configuration solutions until the proposed product meets the requirements.

Figure 6 shows the automaton representing the conversation flow specification of the configuration service. We have labeled the arcs with a boldface identifier to simplify the identification of the conversation turns. The interaction starts with the consumer contacting the provider in order to configure a product. The consumer might be the personal agent of a customer, or a middle agent invoking different product configuration Web Services on behalf of a customer. We omit the description of the initial part of the interaction, which is very similar to that of Figure 3, and we start our description from state 4 of the automaton. If the provider accepts the interaction, a data specification phase starts (*SPECIFY_DATA* arc) that can end in two ways. An unrecoverable error can be generated, e.g., if the set of features selected by the customer cannot be provided in the same product. In alternative, the customization process succeeds and the service continues the interaction by proposing the product (*ProposeProduct*). The consumer may react in three different ways: it may accept the proposal (*Accept*), reject it (*Refusal1*), or suspend the interaction. If the consumer rejects the proposal, another product has to be specified.

The data specification phase includes the provision of the customer's data and of the requirements on the product to be customized. This phase is a complex one and is characterized by the portion of the automaton depicted in the zoom window. When the consumer sets some data, e.g., some product features, the provider may react in different ways. For instance, it may:

- Confirm the correct acquisition of the data ($Confirmation$ arc) and enable another invocation of the $SetData$ operation.
- Notify the consumer about a failure in the product customization process ($Error$) and enable the selection of other values for the conflicting features.
- Suspend the interaction ($SuspendInteraction1$) and resume it later on ($ResumeInteraction1$), in order to manage possible delays in the invocation of its own sub-suppliers.

The $SetData$ operation has a formal parameter ($args$) that is bound to the actual list of features to be set at each interaction step. As mentioned in Section 2 the selection of the feature values

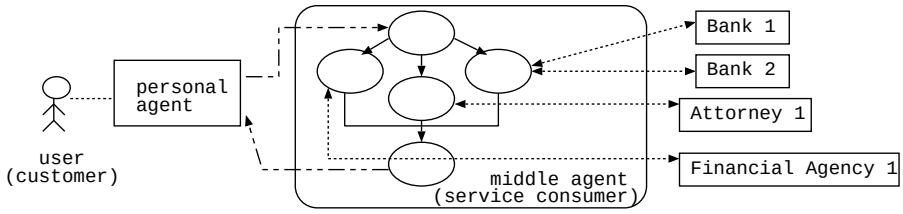

Figure 7: Interaction between a middle agent and some Web Services.

would not be possible without a binding phase, where the consumer analyzes the product structure specified in the Web Service ontology. Sharing the product ontology is thus necessary to let the consumer understand the individual product features to be set.

We have selected the customization of loans as a sample application domain, within the more general customization of products use case. This type of service is particularly interesting because the features of the loans may be customized in rather different ways depending on the customer's characteristics (income, age, etc.) and the destination of the loan (type and features of the good to be funded). With the enhancement of the communication capabilities offered by our conversation framework, the customization of loans could be offered as a Web Service invoked by middle agents that assit customers in the definition of loans by automatizing the contacts with banks, funding agencies and attorneys. Depending on the customer's requirements, the middle agent could invoke different funding agencies and attorneys to propose and manage the organization of a suitable loan. Figure 7 depicts this scenario.

Notice that the possibility of suspending and resuming the interaction offered by our model guarantees that the middle agent and the invoked service providers perform their internal activities without blocking the business interaction in abnormal ways. For instance, if the customer is trying to buy an apartment, the service provider might need to suspend the interaction to evaluate the good. Similarly, after the loan is proposed, the consumer might need to contact the customer to see whether the proposal can be accepted. In both cases, the conversation must be explicitly stopped to let the participants carry other activities out.

4.3 INTERACTION MANAGEMENT

In order to manage the conversation at both sides the participants should run, respectively, a *Conversation Manager* and a *Conversation Client* module. The former is employed by the provider to manage the interaction with the consumers, which would rely on the light Conversation Client to parse the incoming messages and return the responses. The situation is shown

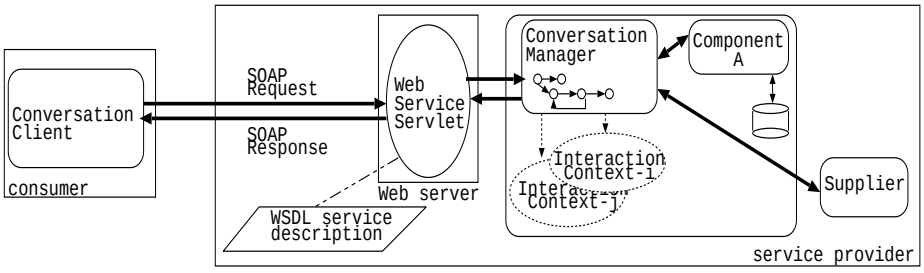

Figure 8: Interaction with a Web Service provider.

in Figure 8, that sketches the architecture of the proposed framework.

The Conversation Manager should exploit a conversation automaton, such as the one depicted in Figure 6, to control the service provider's communicative behavior. For each interaction session, the Conversation Manager of the provider should maintain 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 to support other types of behavior, such as the suspension of a conversation and the subsequent restart. When a consumer starts a conversation ($initInteraction$ arc), the service provider's Conversation Manager should initialize the interaction context by setting the active state of the conversation to the initial state of the automaton (state 2). Then, the provider should choose the continuation path ($Agreement$ or $Refusal$) and move the active state accordingly. During the interaction, the active state should be updated depending on the messages that the provider sends or receives. Each turn is an asynchronous $SendM$ message that one of the participants should send within time constraints. If the the message does not reach the recipient in time, the interaction is suspended by its conversation module.

The Conversation Client has three responsibilities:

- *Facilitating the reception and interpretation of messages at the consumer side.* The Conversation Client manages the reception of messages and extracts the eligible continuations of the interaction from the incoming message (whose $nextOperations$ argument includes the next operations that can be performed and their actual parameters). Given the list of alternatives, the consumer is responsible for choosing the most convenient option and deciding the details of the invocation, depending on its own business logic. For instance, in the data specification phase, the Conversation Client instructs the consumer about the product features that must be set; e.g., the amount of money to be funded, between 0 and 50000.00 EUR. Then, the consumer chooses the values to be set; e.g., 20000.00 EUR.

- *Supporting the correct invocation of the operations on the provider.* The Conversation Client assists the consumer when it sets the actual parameters of the operations to be invoked by performing type checks and other consistency checks aimed at guaranteeing that the parameter values satisfy the existing constraints on the arguments of the operations.

- *Facilitating the management of the outbound messages to the provider.* When the details about the invocation are provided, the Conversation Client generates the invocation of the operation on the provider and sends the $SendM$ message.

The second item above plays a critical role in the enforcement of the correct invocation of the provider. Indeed, the consumer might rely only on the specification of the admissible operations and their actual parameters available as $nextOperations$ information. However, the consumer might make a mistake when binding the arguments of the object-level action to be invoked. For instance, the consumer might invoke the $SetData$ operation by specifying String parameters where Integer ones are needed. Moreover, although the amount of money to be funded has to be between 0.00 EUR and 50000.00 EUR, the consumer might invoke $SetData$ by specifying that it asks for 70000.00 EUR. In these cases, the provider should reply with a failure message ($Error$ arc in the zoom of Figure 6) and the consumer should repair the error by invoking the operation again, with different parameter values.

In order to minimize the message traffic due to constraint violations and type inconsistencies, we propose to enrich the Conversation Client with communication capabilities that enable

the local error management. In this perspective, the Conversation Client becomes a representative of the Web Service to be downloaded by the consumer and interacts with it in decentralized way. The Conversation Client works at the granularity level of the individual parameters of the operations to be invoked, checking the types of the values selected by the consumer and the constraints between the parameters. It should be noticed that, although the local checks support the repair to several problems within the consumer, they cannot prevent the failure of the overall product customization, which might occur, for instance, if the overall set of selected features is inconsistent (i.e., the consumer's requirements cannot be jointly satisfied). This general type of failure is detected at the service provider side and is handled as specified in the *UnrecoverableError* transition of the automaton depicted in Figure 6.

5 INTERACTION MANAGEMENT INFRASTRUCTURE

We are developing a set of Java libraries aimed at facilitating the development of a Conversation Manager and a Conversation Client modules supporting the communication between Web Service providers and their consumers. These libraries support the engagement in long-lasting interactions that may be suspended and resumed depending on the needs of the participants.

A Java-based Conversation Manager module that 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: 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. 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; see Figure 8. 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. Figure 8 shows a situation where the Conversation Manager is handling two parallel interactions and thus maintains two interaction contexts, i and j. 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 FSA shown in Figure 6 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. The Conversation Client catches the messages that the Web Service sends to the consumer and interprets them to identify the eligible operations that may be invoked next. The Client also supports the generation of the *SendM* messages to respond to the Web Service. The Conversation Manager and the Conversation Client exploit the Sun Microsystem Web Service Developer Pack [Sun Microsystems, 2003] and in particular the JAXP-RPC package.

The JWSDP allows the development of Java code that is automatically translated to SOAP messages following the WSDL specification. In our work, we have exploited this feature to support the interoperability of a Java-based Web Service with consumers developed in heterogeneous environments. More specifically, we enable the service provider running our Conversation Manager to handle inbound and outbound messages by applying the SOAP communication

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyConversationService" targetNamespace="urn:Foo"
   xmlns:tns="urn:Foo" xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org..." xmlns:soap="http://schemas.xmlsoap....">
<types>
 <schema targetNamespace="http://java.sun.com/jax-rpc-ri/internal"
         xmlns:tns="http://java.sun.com/jax-rpc-ri/internal" ... ">
  <import namespace= ... "/>
  <complexType name="SendMArgs">
   <sequence>
    <element name="context" type="anyType"/>
    <element name="convId" type="string"/>
    <element name="currentOperation" type="string"/>
    <element name="nextOperations" type="ns2:vector"/>
   </sequence>
  </complexType> ... DEFINITION OF OTHER COMPLEX TYPES ...
 </schema>
</types>
<message name="Conversation_sendM">
 <part name="SendMArgs_1" type="tns:SendMArgs"/>
</message> ... DEFINITION OF OBJECT-LEVEL OPERATIONS; e.g., SetData
<portType name="Conversation">
 <operation name="sendM" parameterOrder="SendMArgs_1">
  <input message="tns:Conversation_sendM"/>
  <output message="tns:Conversation_sendMResponse"/></operation></portType>
<binding name="ConversationBinding" type="tns:Conversation">
 <operation name="sendM">
  <input><soap:body encodingStyle="http://schemas.xmlsoap.org.../></input>
  <output> ... </output> ...
 </operation>
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
</binding>
<service name="MyConversationService"> ... </service>
</definitions>
```

Figure 9: Portion of the WSDL specification of the loan customization service.

protocol to exchange Java-based messages. However, we also provide a declarative representation of the format of the $SendM$ messages, including the WSDL specification of the object-level operations requested by means of the *SendM* messages. Following the WSDL specification, the schema defining datatypes and object-level operations, such as *SetData*, are generated, as well as the ports and bindings needed to interpret WSDL messages. Thus, a generic consumer, which does not exploit our Conversation Client, may invoke the Conversation Manager by conforming to the WSDL specification of its services. Moreover, the consumer may be guided by the provider in the correct invocation of services if it parses the incoming *SendM* messages and extracts the *nextOperations* information. The infrastructure enforces the WS-I standards on complex object type to ensure interoperability between Web Service providers and consumers developed in heterogeneous environments.

From our perspective, the $SendM$ operation is a service offered by both service providers and consumers and must be added to their WSDL specification. In fact, the consumer needs to offer the $SendM$ operation in order to receive messages from the provider. However, the

publication of the *SendM* operation does not impose any overhead because it is automatically generated by our framework. Figure 9 shows a portion of the WSDL declarations generated to support the management of *SendM* messages and includes the specification of the *SetData* object-level operation related to the customization of loans. For readability purposes we have removed some information from the WSDL specification.[2]

6 RELATED WORK

The Semantic Web community is defining standards for the publication of Web Services aimed at overcoming the main limitations of WSDL and of the emerging workflow management standards. The semantic approach differs from the pure XML-based ones because it specifies Web Services at the application layer, describing *"what a service can do, and not just how it does it"* [Web Services Coalition, 2002]. The description of a Web Service specifies the domain ontology underlying the service, the meaning of the operations to be invoked and the service choreography with the major advantage 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; e.g., see [Kamamura et al., 2003]. Some recent work also proposes prototype infrastructures for the run-time conversation management that rely on the semantic representation of 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. 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 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 significant overhead on the interaction management. In our work we are deeply concerned with the scalability and applicability constraints imposed by the Web. For this reason, we try to reduce the complexity added by semantic information as much as possible, and to handle the interaction between service consumer and provider in a lightweight fashion, at least at the consumer side.

The same considerations are useful to relate our work to the Multi-Agent Systems research, where the interaction between distributed processes is regulated by defining coordination protocols such as the FIPA Contract Net [FIPA, 2000], and JAFMAS conversations [Chauhan, 1997]. Unfortunately, these approaches are not directly applicable to the open environment of internet-based applications because they require the agreement on specific communication languages, such as FIPA ACL [FIPA, 2000] and interaction protocols, as done in the AgentCities project [Agentcities, 2002]. Moreover, Web service providers and consumers may have different business logics and the agreement on a conversation policy satisfying both of them is not trivial.

Other XML-based standards for the specification of e-business interactions with Web Services are currently submitted as W3C standards. 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 by the conversation participants. To this purpose, WSCL exploits a sequence diagram model that the participants

[2] See http://www.di.unito.it/~liliana/appendix.txt for the complete representation of this portion of the service.

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 [Hanson et al., 2002]) 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 make the interaction with complex Web Services easier from the consumer application viewpoint.

Our interaction model differs from the previous ones because they assume that each participant maintains an internal record of the conversation state. Instead, we propose that only the service provider handles the conversation state. In particular, we simplify the implementation of the consumer, which is supplied 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 provider, for two reasons. First, the provider has to maintain the context for each interaction session, otherwise, it would not be able to execute the service requests. Second, the provider knows the details of the execution of its own services. Notice that both cpXML and our work aim at decoupling the business logics of service providers and consumers 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 impose the execution of any specific conversation policy. Moreover, our framework differs from WSCL and WSCI because they conform to WSDL in the specification of *request-response* and *solicit-response* operations; thus, they cannot support a fine-grained specification of the conversation turns.

7 CONCLUSIONS

We have presented a conversation model supporting the interaction between Web Service providers and consumers. Our approach is based on the idea that the following factors facilitate the accessibility of Web Services and the establishment of short-term business interactions:

- The decoupling of the business logics by means of the conversational activity.
- A flexible but simple conversation model supporting the exchange of several asynchronous messages during the same interaction.
- The server-side control of the interaction, aimed at minimizing the communication overhead imposed on the consumer. The consumer does not need to know the conversation flow because the interaction is driven by the provider.

Our proposal builds on the speech acts dialog model, that represents communicative behavior as actions performed by an actor towards a recipient. However, our approach is simplified in several aspects in order to address applicability requirements that can seriously affect the usefulness of the conversation model in real cases. For instance, our representation does not support the specification of different types of speech acts, with their semantics. Nevertheless, it clearly separates the conversational activity from the domain specific behavior that is represented as object-level actions the partners "talk about". The explicit representation of the interacting processes and of their conversational activity supports the detailed and unambiguous specification of the possible sequences of turns at the granularity level of the individual messages to be exchanged.

As discussed in [Deo, 2002], the invocation of Web Services should be as seamless as possible. Although the current Web Service specification frameworks address this constraint by

providing stubs that can be run by consumers, we aim at providing interactivity between the providers and consumers. Moreover, we take into account the fact that, during the service fruition, the consumer has to be guided by the provider both in the identification of the operations to be invoked and in the selection of the parameter values. Thus, we propose a framework enables the consumer to locally perform simple type checks and constraint propagation activities, aimed at enforcing the invocation of operations. More specifically, our framework offers:

- The libraries for the specification and implementation of the conversation automaton held by the service provider.
- The libraries needed by the provider to generate the messages guiding the consumer in the invocation of operations.
- A Conversation Client that the consumer can download and run to interpret the service provider's messages and to handle the interaction with the Web Service.

Notice that our conversation model obviously introduces some overhead in the communication, with respect to the basic WSDL approach. In fact, the messages between service provider and consumers are more complex, as they include turn management and parameter binding information. However, our approach has the following advantages: first of all, the definition of operations having generic parameters dramatically reduces the number of WSDL operations that should be published by the Web Service. Second, the dynamic selection of the eligible conversation turns that the consumer may choose from supports the communication with highly interactive Web Services. Third, the local management of type and constraint checks at the consumer side significantly reduces the failures to be handled by the service provider, thus reducing the overall number of messages to be exchanged.

In our future work, we want to investigate the behavioral properties of our conversation model by exploiting formal models for the specification of process dynamics. Our goal is to enable the designer of the Web Service conversation flow to check the correctness of the specified flow against abnormal situations, such as deadlocks, and revise the flow accordingly. We also want to extend our framework to facilitate the integration of conversation and workflow management. In fact, the business logic of a Web Service provider should be managed by employing a workflow engine (such as BPEL4J) that launches and coordinats the processes within the Web Service. In order to flexibly interact with consumers, the engine should invoke the conversation modules, which would take care of managing the interaction context and generating the conversation turns. At the moment, the workflow engine and the conversation modules have to interact with one another as separate programs, but we would like to embed our framework in the workflow engine, so that the conversation steps are managed as sub-processes of the main engine execution.

Finally, our future work includes the management of transactional Web Services. Special operations, such as reservations and payments, have to be implemented as transactions in order to cancel their effects if they cannot be successfully completed. As discussed in [Curbera et al., 2003], internet-based transactions cannot be implemented by exploiting the locking techniques adopted in distributed systems, given the large number of concurrent processes that might interact with the Web Service and its own suppliers. In alternative, [Curbera et al., 2003] and [Benatallah et al., 2003] propose the *compensation* approach to handle the failures in the execution of operations. Roughly speaking, a compensation is carried out by invoking a specialized procedure devoted to restoring the state of the Web Service prior to the

invocation of the failed operation. Our framework can be extended with compensations without major efforts because it explicitly manages failures at the conversation level. Depending on the failed operation, a compensation procedure to be handled at the workflow management level can be invoked.

References

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

[Arkin et al., 2002] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., and Zimek, S. (2002). Web Service Choreography Interface 1.0. http://ifr.sap.com/wsci/specification/wsci-specp10.html.

[Benatallah et al., 2003] Benatallah, B., Casati, F., Toumani, F., and Hamadi, R. (2003). Conceptual modeling of Web Service conversations. In *Proc. Advanced Information Systems Engineering, 15th Int. Conf., CAiSE 2003*, Klagenfurt, Austria.

[Cabrera et al., 2002] 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/.

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

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

[Curbera et al., 2002a] Curbera, F., Duffler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. (2002a). Unraveling the Web Services Web. *IEEE Internet computing*, March-April.

[Curbera et al., 2002b] Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002b). Business process execution language for Web Services, version 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[Curbera et al., 2003] 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).

[Deo, 2002] Deo, H. (2002). The need for a dynamic invocation framework.

[Finin et al., 1995] Finin, T., Labrou, Y., and Mayfield, J. (1995). KQML as an agent communication language. In Bradshaw, J., editor, *Software Agents*. MIT Press, Cambridge.

[FIPA, 2000] FIPA (2000). Foundation for Physical Intelligent Agents. http://www.fipa.org/.

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

[ILOG, 2002] ILOG (2002). ILOG JConfigurator. http://www.ilog.com/products/jconfigurator/.

[Kamamura et al., 2003] Kamamura, T., De Blasio, J., Hasegawa, T., Paolucci, M., and Sycara, K. (2003). Preliminary report of public experiment of semantic service matchmaker with UDDI business registry. In *Proc. Int. Conf. on Service-Oriented computing (ICSOC 2003)*, 208–224, Trento, Italy.

[Mailharro, 1998] Mailharro, D. (1998). A classification and constraint-based framework for configuration. *AI in Engineering, Design and Manufacturing*, 12:383–397.

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

[Paolucci et al., 2003] Paolucci, M., Sycara, K., Nishimura, T., and Srinivasan, N. (2003). Toward a Semantic Web e-commerce. In *Proc. of 6th Int. Conf. on Business Information Systems (BIS'2003)*, Colorado Springs, Colorado.

[Piller and Schaller, 2002] Piller, F. and Schaller, C. (2002). Individualization-based collaborative Customer Relationship Management: motives, structures, and modes of collaboration for Mass Customization and CRM. Technical Report 29, Department of General and Industrial Management, Technische Universität, München.

[Rich et al., 2002] Rich, C., McDonald, D., Lesh, N., and Sidner, C. (2002). COLLAGEN: Java middleware for collaborative agents services with multiple suppliers. http://www.merl.com/projects/collagen.

[Searle, 1975] Searle, J. (1975). Indirect speech acts. In Cole, P. and Morgan, J., editors, *Syntax and Semantics: Speech Acts*, volume 3, 59–82. Academic Press, New York.

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

[Sun Microsystems, 2003] Sun Microsystems Inc. (2003). Java Web Services Development Pack 1.3. http://java.sun.com/webservices/webservicespack.html/.

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

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

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

8 List of Captions

$Figure1$: Service composition and conversation.
$Figure2$: Speech-act based interaction flow for a trivial product customization Web Service.
$Figure3$: Simplified conversation flow specification.
$Figure4$: Domain-level operations and communicative actions.
$Figure5$: Instructing the consumer about how to continue a conversation.
$Figure6$: Conversation flow specification of a product customization Web Service. The portion of automaton in the square is a zoom on the SPECIFY_DATA arc.
$Figure7$: Interaction between a middle agent and some Web Services.
$Figure8$: Interaction with a Web Service provider.
$Figure9$: Portion of the WSDL specification of the loan customization service.