Delivering Semantic Web Services *

Massimo Paolucci
The Robotics Institute,
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA. USA
paolucci@cs.cmu.edu

Katia Sycara
The Robotics Institute,
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA. USA
katia@cs.cmu.edu

Takahiro Kawamura
Research & Development
Center, TOSHIBA Corp
Tokyo, Japan
takahiro@isl.rdc.toshiba.co.jp

ABSTRACT

The growing infrastructure for Web Services assumes a "programmer in the loop" that hardcodes the connections between Web Services and directly programs Web Service composition. Emerging technology based on DAML-S and the Semantic Web allows Web Services to connect and transact automatically with minimal or no intervention from programmers. In this paper we discuss the problems related with autonomous Web Services, and how DAML-S provides the information to solve them. Furthermore, we describe the implementation of two demonstration systems that use such technology: the first system is a B2B application in which a business that assembles computers automatically finds partners providing parts and automatically transacts with them; the second describes an e-commerce application that helps a user to organize a trip to a meeting automatically interacting with different Web Services and the calendar of the user stored in MS Outlook. The results of these experiments show how Web Services can be deployed on the Web to interact and provide information dynamically; second, how the transaction can be carried on automatically with no programmer intervention.

Keywords

Web Services, Semantic Web, Semantic Capability Matching

1. INTRODUCTION

Web services are defining a new paradigm for the Web in which a network of computer programs become the consumers of information. The growing infrastructure for Web Services is based on SOAP [20] and WSDL [2] assumes XML [19] as unifying language to guarantee Web Services interoperability. XML guarantees syntactic interoperability by providing a standard for a common syntax that is shared across the Web, with the result that Web Services can parse each other message, verify whether they adhere to the expected formats, and locate each piece of information within the message. Unfortunately, the two Web Services do not have any means to extract the meaning of the messages exchanged. The two Web Services are in the awkward position of understanding the structure of each other message, but not understanding the content of those messages. The limitation requires programmers to hardcode Web Services with information about their interaction partners, the messages that they exchange and the interpretation of the messages that they receive. The result is a set of rigid Web Services that cannot reconfigure dynamically to adapt to changes without direct human intervention.

Ideally, we would like Web Services to act autonomously, to require the minimal human intervention as possible. Web Services should be able to register autonomously with infrastructure Registries such as UDDI [17], in addition they should use the infrastructure Registries to locate providers of services that they need, and finally, they should be able to transact with these Web Services sending them information formatted in a way that they can understand, and be able to interpret the information that they receive as a response. Autonomous Web Services not only minimize the human intervention by automating interaction with other Web Services, allowing programmers to concentrate on application development, but also they should be able to recover from failures more efficiently by automatically reconfiguring their interaction patterns. For example, if one of their partners is failing or it is becoming unreliable, they may be able to find other more reliable partners, similarly, if a new and cheaper, or anyway better, provider comes on line, Web Services should be able to switch to work with the new provider.

Autonomous Web Services need to be able to find partner Web Services, in order to do that they need to be able to describe and register their own capabilities with public registries, as well as locate other Web Services with specified capabilities. Capability information is crucial for Web Services to locate each other on the bases of the services that they provide rather than on the bases of their name or of the name of the company that deploys the Web Service. In addition, a Web Service should have information on how to interact with the provider, which means that it should know the interaction protocol of the provider, and binding information. Most crucially, this information should allow the requesting Web Services as well as the provider to decode the information exchanged, so it should specify not only the format of the messages to exchange or the remote procedures to call, but also the semantic type of the information to exchange. This view is embraced by DAML-S [16] which defines a DAML [4] ontology for the description of Web Services that attempts to bridge the gap between an infrastructure of Web Services based essentially on WSDL [2] and SOAP [20], and the Semantic Web [1]. In other words, DAML-S bridges the gap between the specification of the format of the information to be exchanged and the specification of its meaning.

DAML-S assumes a view of Web Services that is wildly shared in the community. It assumes that a transaction between Web Services involves at least three parties: a provider of the service, a requester of the service, and some infrastructure component such as UDDI that facilitates the location of the provider and possibly facilitates the transaction between provider and requester. Further-

*(Produces the WWW2003-specific release, location and copyright information). For use with www2003-submission.cls V1.0. Supported by ACM.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ISBN 963-311-355-5.

more, DAML-S allows for a flexible assignment of roles in which a Web Service can be both a the provider in a transaction and a requester in another, and also it allows for a role switch within the same transaction. DAML-S is constructed in three modules that provide a description of different aspects of Web Services. The first one, called Profile, is an abstract description of the Web Service and of the transformation it implements described as a transformation from the inputs the Web Service requires to the outputs it generates. The second module is the Process Model that characterizes the Web Service, specifically, it describes the interaction flow with the Web Service, what function is produced by each step. The third module, called Grounding, specifies how the input/outputs of each step are mapped on WSDL specifications of messages that the two Web Services exchange.

DAML-S provides all the information Web Services need to interact on the Web. DAML-S supports discovery by allowing Web Services to describe their capabilities in the Service Profile so that they can be matched with requests of capabilities. DAML-S capability description and the capability matching[1] [12], extends the UDDI registry[11] allowing Web Services to register their own capabilities and to locate providers of the functionality they seek. Once the provider is located, the requesting agent can use the Process Model and the Grounding to interact with the provider. The Process Model describes the interaction workflow of the provider so the requester can derive what information the provider needs at any given time. Through the Grounding the requester compiles the messages to exchange with the provider.

We tested DAML-S in two applications that stress different aspects of DAML-S. The first system is a B2B application in which a hypothetical computer manufacturer looks for providers of computer parts. The goal of the computer manufacturer is to locate providers of parts of a computer, and negotiate prices of parts and a schedule of delivery. The second application is a variant of travel management application that is often used as use case of application of Web Services. In this latter application the goal of the requester is to locate a travel agent Web Service and book a flight to a meeting while synchronizing the schedule of the flight with the schedule of the user stored in MS Outlook.

In the remaining of the paper we will discuss in detail the architecture of the applications. Specifically, in section 2 we discuss the challenges of automatic composition of Web Services; in section 3 we discuss how DAML-S meets those challenges; in section 4 we discuss the architecture of the Web Service; in section 5 we discuss the computational requirements on a DAML-S Web service; in section 6 we discuss details of the implementation; and finally in section 7 we conclude.

2. COMPOSITION CYCLE

A transaction between Web Services typically involves three or more parties: a requester. one or more providers and a registry, such as UDDI, that supports the Web Services during the transaction and possibly mediates between the requester and the provider. Roughly speaking, the requester corresponds to the client, and the provider corresponds to the server, with the cavia that we expect Web Services to be able to play both roles, for instance a Web Service may be a client in one transaction while a server in another transaction. Furthermore, even within the same transaction the client server relation may switch when the server asks the client to decide between alternatives, or to provide additional information.

Web Services composition follows the cycle described in figure 1 and it can be segmented in two phases phases: the location

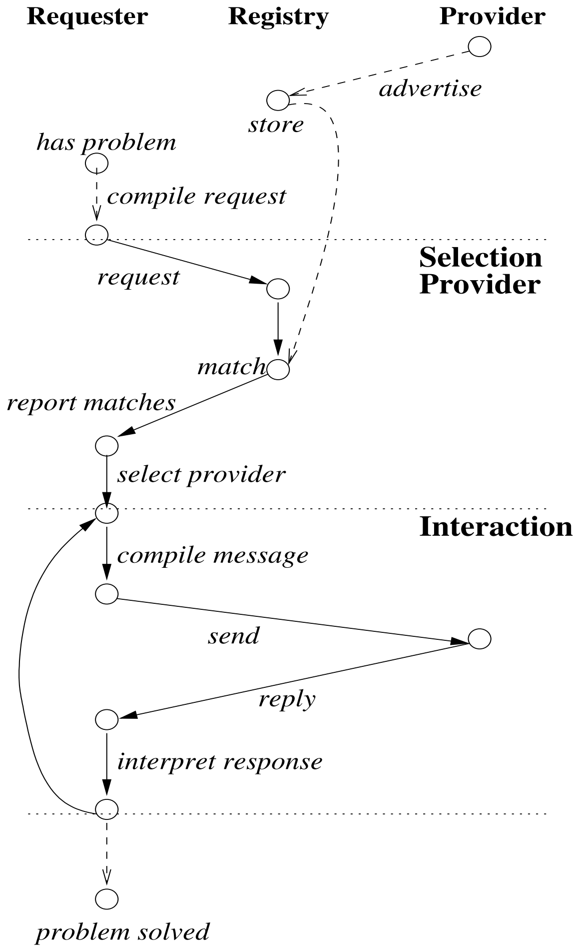

Figure 1: The composition cycle.

of the provider, and the interaction between the requester and the provider. Consistently with registries like UDDI, we assume that whenever a provider comes on line, it advertises with the Registry to make itself known and available to requesters. Strictly speaking, the advertisement of a provider is not a part of a transaction, since the same registration is used in multiple transactions, nevertheless it is an essential precondition for the transaction to take place.

2.1 Location of Providers

The process of locating a provider is composed of three stages, first the requester has to compile a request for a provider and send it to the Registry. Second, the Registry has to match the request with advertisements it stored of Web Services available on the Web, third the requester selects the provider that more closely fits its needs.

The requester at this point knows that problem it expects the provider to solve, but has no idea of what providers are available to solve that problem. In order to find a provider it needs to be able to inquire the registry to locate Web Services with a given capability. The automatic compilation of a request requires an abstraction from the problem the requester faces to the capabilities the requester expects the provider to have in order to solve the problem. Crucially, the solution of the problem also requires an advertisement and query language that supports the representation of capabilities of Web services, so that the Registry receives capability information and processes it.

The task of the registry is to locate an advertisement that matches

[1]A matchmaker is available at www.damlsmm.ri.cmu.edu

a request. The matching of the request and the advertisement should guarantee that the selected Web Service produces the effects that the requester expects. It is crucial to stress that the matching process should take into account that different parties with different perspectives may provide radically different descriptions of the same service. The matching process therefore should not be restricted to an exact match between the request and the provided advertisements, rather it should allow for degree of matching in which Web Services that provide a service similar to the one requested are selected, while Web Services that definitely do not provide the service are discarded. The result is a list of potential providers among which the requester has to select a provider.

The selection of the providers requires a type of registry that is outside the registries offered by the growing Web Services infrastructure, namely UDDI. UDDI stores a host of useful information about Web Services such as information about the organization that fielded the Web Service; binding information to allow Web Services to interact, and an unbounded set of properties, called TModels, that allow to attach any additional information to a Web Service. The problem of UDDI is that it does not have an explicit representation of what the Web Service does. Therefore, the search for a Web Service with a given capability becomes very difficult. As an example, to locate a Web Service that reports weather information within the US, a requester may look for all the Web Services that contain a TModel associated with a classification of services such as NAICS [18] which are specified as weather providers, and all the Web Services descriptions that contain a TModel that associate the Web Service with the US, and then look in the intersection of the results of the two searches. The problem of course is that this type of search cannot distinguish between weather services that provide information about the US, from US based weather services that may provide information about the weather in other countries. Overall, because UDDI misses any form of capability representation and capability matching, it is extremely difficult to find Web Services with a desired capability using UDDI.

2.2 Selection of Provider

The result of the matching between the request and the advertisements will result in a number of matches among which the requester will have to select the provider it wants to use. In general there is no hard and fast rule for the selection of the provider, and it soon turns out to be a domain specific decision.

The simplest thing is to select the provider with the highest score among the Web Services reported by the Registry. A more general approach could be based on decision theoretic reasoning, in which the requester selects the provider that maximizes some utility function, but, in practice, it is unlikely that Web Services will make use of an explicit utility model that they can leverage on.

Other types of information that are not contained neither in the request nor in the advertisement such as the credit history of a provider, or the amount of work the requester estimates to do in order to satisfy all the inputs and preconditions expected by the provider and in general the likelihood that the requester can gather all the input information and achieve all the preconditions expected by the provider.

The strategies proposed above assume that the requester will select only one provider to interact with. This decision may be extremely risky and inefficient in the world of Web Services. One nice part of Web Services is that they run on the same time on different machines implementing actual parallelism. As a consequence, the requester may attempt to interact with multiple providers in the same time; with the advantage that by delaying the selection of the provider it reduces the risks of an early decision. Of course, the problem of this strategy is that the requester may commit to buy the same service from multiple providers, incurring in additional costs at no gain [5]. Therefore, this strategy can be followed only when the requester is aware of the place in the transaction where it commits to buy, at which point the requester may commit on a provider. For example, a requester interested in buying books may try to interact with more than one book selling Web Service in the attempt to find the cheaper one. This will be a good strategy as long as the requester selects from which Web Service to buy before it commits to buy the same book from all the providers.

2.3 Interaction with Provider

Once the provider has been selected the requester should initiate the transaction with it. To support the interaction the provider has to make public its own interaction protocol declaring what information it expects from the requester, in what order, the format of such information and binding information that specify the ports the provider and the requester will use during the transaction. Still, the declaration of the sequence of messages is not enough for the provider. Rather it needs also to declare the material consequences of each step in the protocol.

Current Web Services standards address different parts of the problem. Protocols like XLang [15], WSFL [10] and more recently BPEL4WS [3] address the problem of describing the temporal part of the interaction protocol by specifying workflow models that describe the sequence of messages to be exchanged. WSDL [2] maps descriptions of abstract information to be exchanged by Web Services into message formats and binding that specify where the message is delivered and the transmission protocol.

The specification of the interact protocol is not really enough to allow a successful transaction, rather the requester should derive from the interaction protocol what the provider does with the information it receives. The discussion about commitments in the previous section provides an argument in favor of this requirement: in order to compute its commitments the requester should understand what are the consequences of the steps in the protocol, so the requester will avoid expensive commitments that it does not need. Expressing the consequences of the transaction is not only a problem for parallel interaction with multiple Web Services, rather it is a problem even when there is a single provider and a single requester. For example, a requester will want to make sure that the goods will be delivered after it provides payment information, or that the money will be refunded. These types of inferences are impossible unless the provider specifies also what are the consequences of the message exchange with the requester.

3. IMPLEMENTING WEB SERVICES WITH DAML-S

The analysis of the composition cycle we did above led us to establish what we expect from a language that describes Web Services. The first requirement is that the language supports the description of capabilities of Web Services, or in other words a description of what function the Web Service accomplishes. Furthermore, there should be an algorithm for indexing descriptions and for comparing them to infer whether they describe the same function. The second requirement is that the language allows the specification of the interaction protocol of the provider, but also it should support the specification of the actual consequences of each interaction. Lastly, it should support mapping of abstract information about inputs expected by the provider and outputs generated to a scheme of information transfer that allows the requester and the provider to exchange actual messages. When these requirements are satisfied, it should be possible to construct an infrastructure for Web Services that is centered around a Registry that performs capability matching. Once Web Services find each other, the infrastructure provides sufficient information for them to interact.

DAML-S and the growing infrastructure around it attempts to satisfy all the requirements described above. DAML-S is emerging as a Web Services description language that enriches Web Services descriptions based on WSDL with semantic information from DAML ontologies and the Semantic Web. DAML-S is organized in three modules, the first one is a Profile that describes capabilities of Web Services as well as additional features that help to describe the service. The second module of DAML-S describes the Process Model of the Web Service, specifically it provides a description of the activity of the Web Service provider from which the Web Service requester can derive the interaction protocol and the consequences of each message exchange. The third feature of DAML-S is the Grounding: a description of how abstract information exchanges described in the Process Model is mapped onto actual messages that the provider and the requester exchange.

3.1 Service Profile

The role of the DAML-S Profile is to describe the Web Service capabilities, as well as additional features of Web Services that provides a rich description of the Web Service. DAML-S describes capabilities of Web Services as functions that produce a transformation. This transformation happens at two levels: at the information level a set of inputs are transformed in a set of outputs; at a more concrete level a set of conditions become true, while others become false. For example, if we consider a travel booking Web Service, at the information level it may require departure and arrival information and provides and using that it generates a flight schedule and a confirmation number; while at a more concrete level it books a flight, generate a ticket, and charges a credit card so that the money available on the account is reduced. Capabilities in DAML-S are represented at both levels, at the information level they are represented by the inputs that they require and the outputs that they generate, at the state level by the preconditions for the Web Service to execute and the effects that the Web Service generates.

Since different Web Services with very different features may have the same capabilities, DAML-S allows the specification of a host of additional information about the Web Service that may help during the selection process. These additional information consists of the category of the Web Service as described in the classification of the service within some classification schema, parameters that restrict the use of the Web Service, and quality rating to specify how good is the Web Service provided.

While DAML-S is just a Web Services representation and therefore does not imply any form of processing, it is relatively easy to implement a matching algorithm to recognize which Web Services advertisements match a given request. There is at least one such matching engine [12] that takes advantage of the underlying DAML logic to infer the the logic relations between the input and outputs of the request, with the input and outputs of the advertisements. While a complete description of this algorithm is outside the scope of this paper, the main idea is that a the outputs of the request should be subsumed by the outputs of the selected advertisements, this condition guarantees that the selected Web Services provide the expected information. Furthermore, the matching engine ranks the advertisements on the bases of their input matching, where, inputs match if inputs of the request subsume the inputs of the advertisement. This condition selects services that the requester has enough information to invoke.

DAML-S Profiles play a role that is very similar to the role of UDDI entries in the UDDI registry. Both data structures provide a description of Web Services; indeed it is possible to construct a mapping between DAML-S records and UDDI [11] using TModels to encode the capability information. Once the capabilities are encoded in UDDI, a matching engine based on the DAML-S matching algorithm can be used to retrieve Web Services from UDDI on the bases of their capabilities. An initial version of such a registry has been implemented and it is used in the experiments described below.

3.2 Process Model

The Process Model fulfills two tasks, the first one is to specify the interaction protocol in the sense that it allows the requester to know what information to send to the provider and what information will be sent by the provider at a given time during the transaction. In addition, to the extent that the provider makes public its own processes, it allows the client to know what the provider does with the information.

A Process Model is defined as an ordered collection of processes. The DAML-S Process Model distinguishes between two types of processes: composite processes and atomic processes. Atomic processes correspond to operations that the provider can perform directly. Composite processes are used to describe collections of processes (either atomic, or composite) organized on the basis of some control flow structure. For example, a sequence of processes is defined as a composite process of type sequence. Similarly, a conditional statement (or *choice* as defined in DAML-S) is also a composite process. The DAML-S process model allows any type of control flow structure including loops, sequences, conditionals, non-deterministic choice and concurrency. Because of its expressivity, the DAML-S Process Model can be used to represent any arbitrary workflow.

Processes are defined as transformations between an initial state and a final state. The initial state is defined by the inputs of the process and a set of preconditions for the process to run successfully. Inputs represent the information that the Web Service needs to collect to execute the process correctly; while preconditions represent conditions that have to be true for the execution of the process to succeed. The result of a process is described as a set of outputs, or information that results from the execution of the process, and a set of effects that represent physical changes that result from the execution of the process. DAML-S distinguishes two types of input and outputs: the first type are internal input and outputs, in such a case the output of one process will feed into the input of a following process. The second type are external input and outputs: they define information that will be provided by a requester, and that will be reported to a requester.

During the interaction with the provider, the requester analyzes the process model to infer what process the provider is currently executing. The requester is particularly interested in the input the provider needs and the outputs that result from the execution of the process, since most likely the requester will have to provide the input information and interpret the output information.

By following the process model, and interpreting the information received by the provider, the requester can infer what information the provider expects at that time, or what information that provider will send next. Implicitly, the process model of the provider specifies the interaction protocol between the provider and the requester providing details of what information the provider needs and in what order. The message format and the binding information is instead specified by the DAML-S Grounding.

The specification of preconditions and effects of processes allow the specification of the consequences of their execution. The

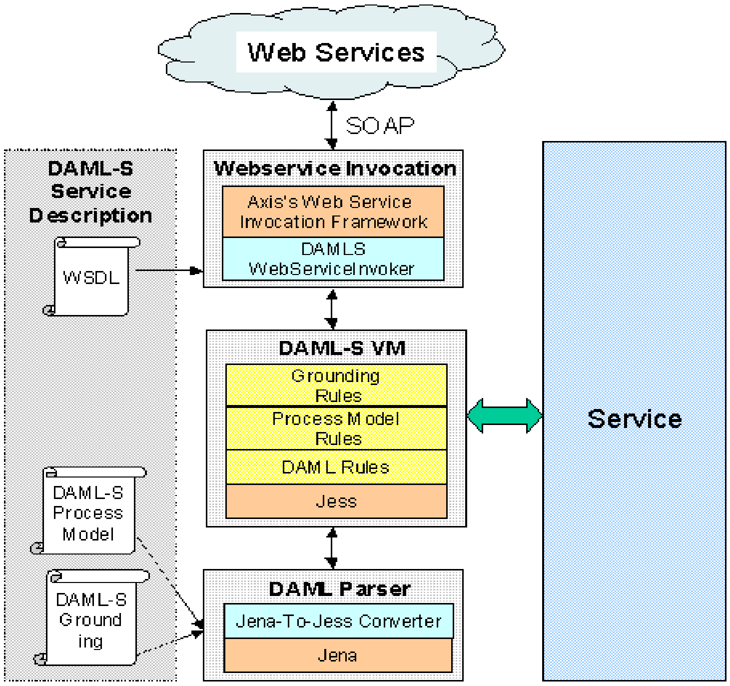

Figure 2: Description of DAML-S Web Service architecture

preconditions specify under what conditions a process can be executed, while the effects specify what results from the execution of the process. As an example, the precondition to a buy action is that the credit card used is a valid one and that it is not overdrawn, while the effect is that the credit card is charged. The role of preconditions is to provide a way to the requester to reduce the likelihood of failures: the requester knows that if the preconditions are not satisfied, the provider will not be able to execute correctly the process. The description of the effects of the process specify the consequences of the execution of the process, so for instance a buying process will have as a consequence the transferring of ownership of some goods and the requester know that after the execution of the process it will own the goods.

Through the specification of input, outputs, preconditions and effects of processes, the DAML-S process model provides the information the requester needs to interact with the provider: specifically, it specifies the interaction protocol with the provider by specifying the abstract messages the the provider and the requester exchange, on the other side it also provides information on the requirements to be satisfied by the requester to execute the Web Service correctly, and what will result from the execution of the processes. It will be up to the requester to make use of this information to decide whether to use a given provider, knowing what it requires and what are the consequences of this choice.

3.3 DAML-S Grounding

The DAML-S Grounding transforms the abstract description of the information exchanges between the provider and the requester into messages that can be asynchronously exchanged, or through procedure call. Specifically, the DAML-S Grounding is defined as a one to one mapping from atomic processes to WSDL specifications of messages. From WSDL it inherits the definition of abstract message and binding, while the information that is used to compose the messages is extracted by the execution of the process model.

The integration of WSDL in the DAML-S specification facilitates the interaction between non-DAML-S Web Services with Web Services that rely on DAML-S to describe the workings. From a more theoretical view point, it describes the position of DAML-S within the growing Web Services infrastructure through the specification of the role is played by DAML-S and what role is played by WSDL.

4. A WEB SERVICE ARCHITECTURE

The discussion above shows that DAML-S provides a promising framework to control Web Services interaction. It provides all the information that Web Services need to negotiate autonomously on the Internet, minimizing interventions of programmers while maintaining a very flexible and reliable connection with their providers. Still, it leaves open the problem of harvesting all the information available and make use of it in real implemented Web Services. This is the center of our work as we are trying to develop a Web Service architecture that can take advantage of the information available in DAML-S descriptions and to produce a DAML-S toolkit that facilitates the implementation of DAML-S enabled Web Services.

The first requirement of a DAML-S enabled Web Service is to understand the structure and the information contained in a DAML-S description. This requirement entails the ability of compiling correct requests for service, processing the descriptions of the services retrieved and finally, correctly following the workflow described in the process model as well as drawing the correct inferences on the preconditions and effects of the processes in the Process Model.

A precondition for the first requirement is that Web Services un-

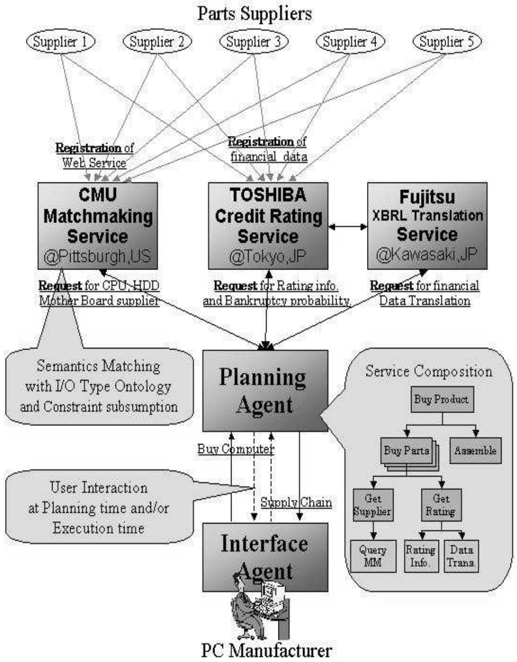

Figure 3: Description of the system for scheduling a trip to the DAML PI meeting

derstand ontologies encoded in DAML. Dealing with DAML ontologies is fundamental because it allows the Web Service to derive inferences on the different statements included in a DAML-S description. Furthermore, it is required to extract the information included in the messages exchanged with other Web Services, as long as this information is compiled in DAML

Finally, the Web Service needs to include ways to use WSDL descriptions of the Grounding to send and receive messages correctly. This last module should be responsible for the exchange of messages with other Web Services managing not only the WSDL description, but also managing the network protocol and the transmission of information or the reception of such information.

The architecture that we adopted is displayed in figure 2. It shows that a DAML-S Web service can be divided in two main parts: a *DAML-S Port* which corresponds to the three modules in the center column, which is responsible for the management of the interaction with other Web services; and the actual *Service*, represented by the box on the right, which determines what the Web service does. Examples of services may include a stock reporting Web service, in which case the Service module would monitor the stock market, or an airline booking Web service, the Service module interacts with the airlines to book seats, gather information about airplanes and so on. Ultimately, DAML-S is silent about the application of the Web service, so we display it as a black box.

The DAML-S Port consists of three modules which are roughly used in a sequence. The *DAML Parser* is used to load DAML-S specifications of Web services, specifically their Process Model and Grounding, as well as loading other DAML ontologies from the Web. The DAML Parser transforms the DAML files into lists of predicates to be processed by the DAML-S Virtual Machine (*DAML-S VM*). The DAML-S VM defines a knowledge base that is based on the Jess Theorem [8], but it specializes with rules that implement the DAML axiomatic semantics [9, 7] as well as the semantics of the DAML-S Process Model and the semantics of the Grounding. Finally, the *Web service Invocation* module takes responsibility for transforming abstract information to the sent to other Web services, into concrete messages, or RPC calls, to be exchanged[2]. During the interaction, Web services exchange information through the Web service invocation module, incoming messages are translated into DAML through the Grounding Rules and added to the Knowledge Base where the new knowledge can interact with the rest of the knowledge of the Web service.

The DAML-S VM is the core of the architecture and it controls the interaction with other Web services. The first task of the DAML-S VM is to follow the Process Model of the Provider, this is accomplished through the implementation of the Process Model semantic with the Process Model Rules. These rules set contains rules the location of the next process to execute, for the extraction of inputs and outputs of each process and for the management of non-deterministic choices. Similarly, the Grounding Rules specify the correspondence between atomic processes of the Process Model and the WSDL operations, and the mapping between the inputs and outputs of the atomic processes in the corresponding input and outputs of the process model.

5. REQUIREMENTS ON THE WEB SERVICE

As described above, DAML-S is mute about the application, therefore any type of application could in principle take this place. Nevertheless, the application level is responsible for many of the

[2]In the picture we refer to SOAP to format messages, in the implementation the message formatting depends on the WSDL specification loaded by the Web service.

decisions that have to be made while using DAML-S. For instance, the application level is responsible for the use of the information extracted from the messages received from other Web Services or to decide what information to send to other Web Services. In order to take advantage of the flexibility supported by DAML-S, the application level should support a decision system that makes non-deterministic choices while maintaining efficiency and control on the behavior of the Web Service.

The Service is also responsible of the non-deterministic decisions that have to be made during the interaction with other Web services. The DAML-S Process Model describes a workflow that may contain conditionals as well as non-deterministic choices. The DAML-S VM can find these selection points, but it is up to the Service to decide which choices to make in those situations; since they requires the Service to analyze which branch of a choice would lead to the goals it wants to achieve. It is therefore essential that the Service include a decision system that can make non-deterministic choices while maintaining efficiency and control on the behavior of the Web Service.

In addition, the Service is also responsible for Web services composition during the solution of a problem. Specifically, the Service module is responsible for the decision of what goals to subcontract to other Web services, and as a consequence of compiling capability descriptions of potential providers to submit to a DAML-S/UDDI Registry; furthermore, it is responsible of the selection of the most appropriate provider among the providers located by the Registry. As a consequence, the neat and modular picture shown in Figure 2 is only partially true, a Service that wants to take advantage of the DAML-S VM should also have access to its DAML inference layer and possibly to the whole DAML-S virtual machine to reason about the Process Model, decide how to deal with non-deterministic decisions or how to react to unexpected decisions of the other parties.

Ultimately, the programmer has two choices, either hardcodes many of the decisions that the Web services has to make during its interaction with other Web services, or it employs a computational mechanism that supports the non-deterministic decision making that the Web service requires. In our implementation, we followed the second path, we employed the RETSINA planner [13] to control the application level. The RETSINA planner is based on the HTN planning paradigm [6] which provides a reliable as well as efficient planning scheme. The advantage of HTN planning over other planning schemes is that HTN plans by decomposition introducing set of actions in the plan where other planners would introduce only one action at a time. Furthermore, while HTN planning has in principle the same complexity of planners from first principles, in practice it reduces the number of decisions that the planner has to make resulting in an overarching efficiency gain.

The RETSINA planner extends HTN planning by adding interleaving of planning and execution which basically allows the Web Service to execute before a plan is completely formed. Interleaving of planning and execution has a number of advantages over traditional planning. For one it allows the Web Service to discover what providers are available and plan its course of actions as a function of those features. The second advantage is that it allows to replan to react to unexpected situations. For instance, if one of the providers fails to respond, the requester may look for an alternative provider to interact with.

6. EVALUATION AND IMPLEMENTATION

To test our approach to DAML-S we implemented two systems with very different characteristics that take advantage of DAML-S and the Web Service architecture described above. The first one is a B2B application in which a Web Service looks for business partners and automatically negotiate business agreements with them. The second application is a B2C application in the travel domain in which a Web Service that functions as personal assistant of a user organizes a trip to a conference by booking a trip to the conference verifying availabilities with the user schedule stored on MS Outlook.

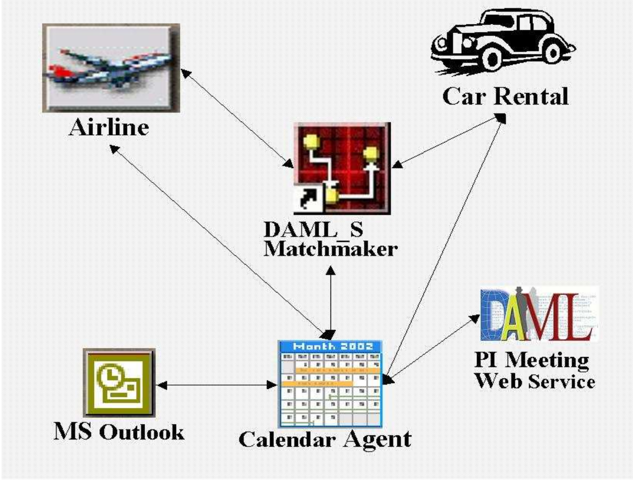

Figure 4: Description of the system for scheduling a trip to the DAML PI meeting

The first system describes a B2B application in which a Web Service that is given the task of assembling computers looks for providers of computer parts. The architecture of the Web Service is described in figure 3. The Interface Agent provides an operator with a way to interact with the planning agent and to compare options of combinations of business partners schedules costs and so on. The Planning Agent employs the planning scheme described above to achieve the goals proposed by the operator, in our case to find providers of computer parts and organize a supply chain that meets cost and time limitations. To achieve its goals the planning agent queries the Matchmaker for potential parts suppliers, then it uses the Toshiba and Fujitsu financial services to verify the likelihood that the suppliers will not bankrupt during production time affecting the whole business. Finally, the planning agent contacts the suppliers to negotiate schedule and costs.

The challenge of this system is to support interaction between Web Services provided by very different organizations geographically spread which we could orchestrate only on the bases of DAML-S information. Furthermore, it allowed us to experiment with suppliers selection using information that is erogenous to DAML-S. It is unfeasible to expect that DAML-S Profiles will contain all the information that the requesters will ever need about the provider, in this case the requester, i.e. the Planning Agent, uses two financial services to gather information about its providers before contacting them.

The Matchmaker used in the system is a DAML-S enhanced UDDI, it uses a freely available UDDI server [3] to store DAML-S advertisements using the encoding described in [11] allowing for capability matching in UDDI.

The organization of the second example is displayed in figure 4; the goal is to book a trip to a conference, namely the DAML

[3]We used initially the IBM test UDDI site, then we switched to the Systinet UDDI server

PI meeting. We assume that the organizers of the meeting publish a Web Service which provides information about the meeting, such as time, location, talks, participants and so on. Through the RETSINA Calendar Agent [14], the user plans a trip to the conference. The Calendar Agent verifies availability checking on the schedule of the user stored in MS Outlook, and then uses the same Matchmaker used in the previous system find airlines, car rental companies and hotels. Finally, uploads the schedule of the trip in Outlook.

This example extends the previous one by using complex process models that we used to implement the Web Service as well as to be loaded dynamically to control the interaction on the client side.

7. CONCLUSIONS

In this paper we outlined the different challenges faced by autonomous Web Services. Furthermore, we showed how DAML-S tackles these challenges by providing the information that allows Web Services to connect and interact autonomously with little intervention from the programmers. Finally, we propose a Web Service architecture that takes advantage of DAML-S information to support automatic discovery and interaction between Web Services.

In our presentation and implementations we concentrated on the feasibility of using DAML-S for the interaction with Web services. While we showed that indeed it is possible to use DAML-S to control such interaction, we also show that to take full advantage of the power of DAML-S the Web service needs to incorporate a computational model that supports non-deterministic reasoning such as the HITAP planner. It is still an open question what compromises we need to make to lower the computational requirements and proceed with a simpler computational model.

In the demonstrations we presented, we naïvely assumed that Web services will negotiate agreements and strike deals. Ultimately, we assumed a world of perfectly honest Web services that deliver every time they receive an order, and that pay every time they receive goods. Such an ideal world does not correspond to reality. Ultimately, we will need to introduce notions like commitments and contracts to back the transaction with legal guarantees. Preconditions and effects of processes can be used to express contracts and commitments, for instance a precondition to the execution of a process is that may be that the receiver signs a contract, while the effect is a commitment to deliver goods by an established date. We are currently investigating these problems.

8. ACKNOWLEDGMENTS

We thank Anupriya Ankolekar, Takuya Nishimura, Terry Payne, Rahul Sing and Naveen Srinivasan for the useful discussions and inputs during the planning and implementation of this work. The research was funded by the Defense Advanced Research Projects Agency as part of the DARPA Agent Markup Language (DAML) program under Air Force Research Laboratory contract F30601-00-2-0592 to Carnegie Mellon University.

9. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[3] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. BPEL4WS White Paper. http://www-3.ibm.com/software/solutions/ webservices, 2002.

[4] DAML Joint Committee. DAML+OIL (March 2001) Language. http://www.daml.org/2001/03/daml+oil-index.html, 2001.

[5] G. Economou, M. Paolucci, M. Tsvetovat, and K. Sycara. Interaction without commitments: An initial approach. In *Agents 2001*, 2001.

[6] K. Erol, J. Hendler, and D. S. Nau. Htn planning: Complexity and expressivity. In *AAAI94*, Seattle, 1994.

[7] R. Fikes and D. L. McGuinness. An axiomatic semantics for rdf, rdf schema, and daml+oil. Technical Report KSL Technical Report KSL-01-01, Knowledge Systems Laboratory Stanford University, 2001.

[8] E. Friedman-Hill. Jess: Java Expert System Shell.

[9] J. Kopena and W. C. Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web.

[10] F. Leymann. WSFL White Paper. http://www-3.ibm.com/software/solutions/ webservices, 2001.

[11] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Importing the semantic web in uddi. In *Proceedings of E-Services and the Semantic Web Workshop*, 2002.

[12] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *ISWC2002*, 2002.

[13] M. Paolucci, O. Shehory, and K. Sycara. Execution in a multiagent team planning environment. *Electronic Transactions of Artificial Intelligence*. forthcoming.

[14] T. R. Payne, R. Singh, and K. Sycara. Calendar agents on the semantic web. *IEEE Intelligent Systems*, 17(3):84–86, 2002.

[15] S. Thatte. XLANG White Paper. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001.

[16] The DAML Services Coalition. DAML-S: Web Service Description for the Semantic Web. In *ISWC2002*, 2002.

[17] UDDI. The UDDI Technical White Paper. http://www.uddi.org/, 2000.

[18] US Census Bureau. North American Industry Classification System (NAICS). http://www.census.gov/epcd/www/naics.html, 1997.

[19] W3C. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

[20] W3C. SOAP Version 1.2, W3C Working Draft 17 December 2001. http://www.w3.org/TR/2001/WD-soap12-part0-20011217/, 2001.