Toward Semantic Web Services

Massimo Paolucci, Katia Sycara, Takuya Nishimura, Naveen Srinivasan
Carnegie Mellon University
Pittsburgh, PA, USA
{paolucci, katia,nishi,naveen}@cs.cmu.edu

Abstract

The Web is moving from being a collection of pages toward a collection of services that interoperate through the Internet. In this paper we show how ontological information improves on the growing Web services infrastructure by adding capability matching and a high degree of autonomy to web services so that they can automatically adapt to changing situations.

KEYWORDS: DAML-S, Semantic Web-enabled Web Services, Web services discovery

Introduction

Web services provide a new model for the Web in which sites exchange dynamic information on demand. This change is especially important for the e-business community, because it provides an opportunity to conduct business faster and more efficiently. Specifically, the Web services infrastructure provides a high degree of interoperability across platforms and operating systems that allows different parties to communicate and interact seamlessly. As a result, Web services provide a unique tool to monitor and manage directly the supply chain to predict and possibly prevent problems that may affect the business development. Ideally, Web services can take up a more active role in the management of the supply chain, indeed they can monitor potential providers and eventually suggest how to modify the supply chain to take advantage of new market conditions such as cheaper providers.

In order to deliver on this promise Web services are asked to act autonomously with minimal human intervention, which challenges the Web services community to develop an infrastructure that first provides registries that allow the automatic location of Web services on the bases of the capabilities that they provide; second, it should go beyond the message exchange between Web services by allowing them to also understand the messages that they adding a level of semantic interoperability.

In the recent years, Web interoperability hinged on the promises of XML to provide a standard for a common language that is shared across the Web. Unfortunately, XML defines only the syntax of such a language, allowing only syntactic interoperation, but failing to provide semantic interoperation. The result is that identical XML descriptions may mean very different things depending on when and who uses them. The lack of XML semantics proves to be an obstacle for the development of Web services that can autonomously act on the electronic market. The limitations of XML encoded information allow Web services to parse each other message and verify whether it adheres to the expected formats, and eventually locate each piece of information within the message. Unfortunately, the two Web Services do not have any means to decode the meaning of the messages exchanged to extract the information contained. The two Web Services are in the awkward position of understanding the structure of each other message, but not understanding the content of such messages.

The limitations of representation entailed in XML is reflected by the growing infrastructure for Web services based standards such as SOAP [17], WSDL [2], UDDI [15] and BPEL4WS [4]. These requires programmers to

hardcode Web Services with information about their interaction partners, the messages to 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. Such Web Services are hardcoded to work with a definite set of providers and cannot modify their pattern of interaction when a new provider comes on line that is better or cheaper. Similarly, they cannot react to problems of their providers: when a Web Service goes off line, the whole supply chain is affected because the Web Services that constitute the nodes of the chain cannot look for alternative providers.

The limitations of the Web services infrastructure can be overcame by providing a semantic markup to the descriptions of Web services to take advantage of the information available on the Semantic Web [1]. The contribution of the Semantic Web is twofold, first, it provides ontologies that act like shared knowledge bases across the Web; second, it provides a logic to infer how such terms combine to form complex concepts and how do they interact with the knowledge already accumulated by the agents. From the point of view of Web services, ontologies function as universal dictionaries so that all Web services share the same interpretation of the terms contained in the messages that they exchange, and by derivation of the whole messages. Furthermore, ontologies provide the bases for the description of capabilities of Web services that cannot be expressed using plain XML nor by any of the Web services standards.

DAML-S [6] assumes the task of bridging the gap between the Web services infrastructure based on WSDL and the Semantic Web. Specifically, DAML-S defines a DAML [5] ontology for the description of Web Services which provides the definition of three different aspects of Web services. The first aspect is the description of the capabilities of the Web service to specify what service is provided. The second aspect is the specification of how the Web service accomplishes its task to specify in details how the service achieve its goals, and what are the requirements on potential requesters that want to interact with it. The last aspect of DAML-S includes a specification of how the information exchanged by the Web service and it requesters maps into actual the messages exchanged by the different parties.

DAML-S provides only the specification of Semantic Web services, still this specification needs to be matched by the implementation of tools for the construction of actual Web services that adhere to the DAML-S specifications. In this paper, we discuss the implementation of such tools. Specifically, we discuss the implementation of the DAML-S Matchmaker, a Web services registry that enhances the UDDI registry with matching of capabilities of Web services to allow the location of Web services on the bases of what they provide rather than their name, ports or other contingent information. Furthermore, we discuss the implementation of a DAML-S Virtual Machine that allows the autonomous interaction and invocation of Web services based only on the DAML-S specification with no need of the prior hardcoding that would be needed by Web services specifications such as WSDL or BPEL4WS.

The rest of the paper is organized as follows: first we will discuss the problem of capability matching, our solution and the application to UDDI. The result of the matching process is a set of potential providers that solve problems for a requesting agent. The task of the requesting agent is to contact these providers and interact with them. The second part of the paper discussed the interaction process between the requester and the providers, and how such as process can be automatically controlled using DAML-S. Finally, we will conclude with an example that shows how all this machinery works together.

A Capability-based Registry

Web services are intrinsically social software artifacts, they do not exist in isolation, rather their work depends on the interaction with other Web services. Figure 1 describes a typical protocol of interaction between Web services. It involves three parties: a *provider* of the service, a *requester* of the service and a *registry* such as UDDI that mediates between the provider and the requester.

The interaction between the three parties can be roughly divided into four different phases.

1. Upon coming on line Web services *advertise* their capabilities (the services that they provide) with the registry. The registry in turn stores the registration for future use.

2. When a requester Web service decides to subcontract the solution of one of its problems to other Web services, it *compiles a request* for the registry describing the type of service it requires. Crucially, the requesting Web service has no knowledge of the services available on line, therefore the only information that can be used to compile the request is the problem itself.

3. The registry compares the request with the advertisements it received to locate those Web services on line whose capabilities *match* the capabilities expected by the requester.

4. Finally, the requester selects the provider that best fits its needs and begin the interaction with it.

The interaction described above follows the same interaction pattern that that is usually accepted for Web services. Indeed, we could use UDDI in place of the registry. What is often overlooked is the crucial importance of describing capabilities, rather than contingent information about Web services such as their name or port in advertisements and requests. From the point of view of the requester, capabilities descriptions allow the specification of what it expects from the provider; while from the point of view of the provider, capability descriptions allow specification of what kind of problems it solves.

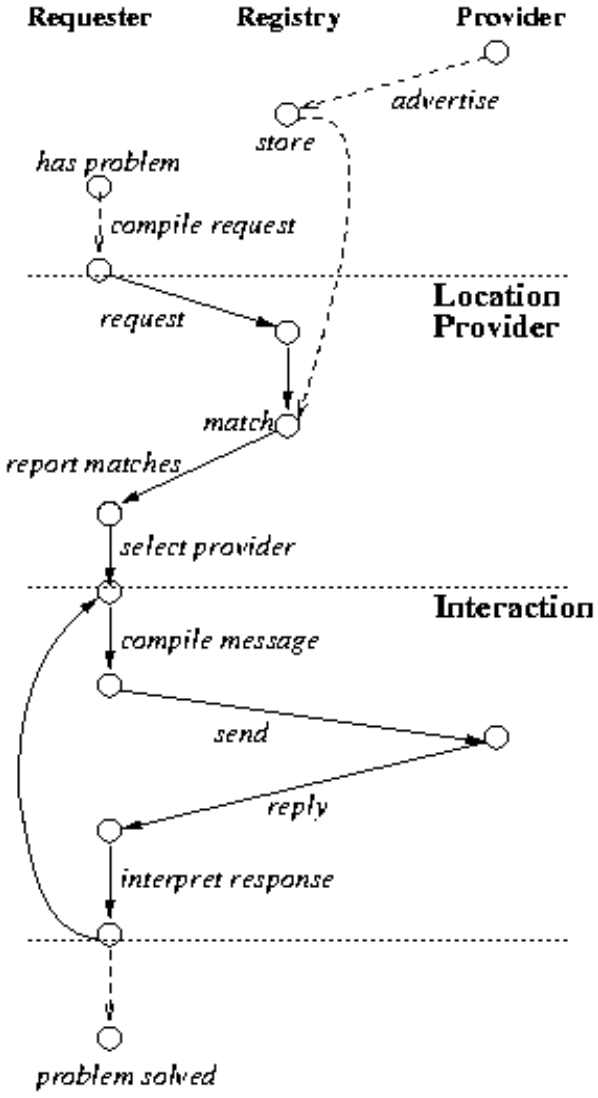

Figure 1: Web Services Interaction Protocol

Specification of capabilities in DAML-S

As mentioned above, DAML-S is characterized by three modules: a Service Profile that describes the Web service and its capabilities, a Process Model that describes in details what the Web service does and how to interact with it, and a Service Grounding that maps the information exchanges described in the Process Model into actual messages between Web services.

DAML-S Service Profiles (hereafter just Profiles) describe Web services from different points of view. Firstly DAML-S describes the provider of the service in terms of who is responsible for the services and who to contact in case the services produced an unexpected behavior. In addition, the Profile provides a host of non-functional parameters that describe general properties of the Web service such as quality guarantees that it can provide, or speed of deliver. The last and most important part of Profiles is to describe capabilities of Web services.

DAML-S assumes a functional view of Web services: a Web service requires some input to perform its task, and generates some outputs as a result. Furthermore, a Web service will function correctly only when some conditions (the service preconditions) are satisfied in the World, and as a result of the Web service execution some effects emerge. Consider for example a book buying service. It requires as precondition that the buyer has a credit card, and as input the credit card number and information about the book the buyer wants. The effect of the Web service is that the buyer owns the book and the output is a receipt that attests the ownership change.

In addition to describing Web services advertisements, Profiles also describe Web services requests for service, i.e. the expectations of the requester. The description is equivalent to the description of the advertisement with the only difference that requests describe a hypothetical Web service with whom the requester would like to interact, rather than a real one.

Algorithm for capability matching

An advertisement and a request match when the Web service advertised provides a service “sufficiently similar” to the service needed by the requester. In its strongest interpretation, an advertisement and a request are ``sufficiently similar" when they describe exactly the same service. Unfortunately, this case is very unlikely; since advertisements

and requests are authored by different parties with very different objectives, and without any a priory agreement, it is natural to expect some degree of mismatch.

Since exact matches are unlikely, the first requirement on the matching algorithm is to accommodate *flexible* matches, i.e. matches that recognize the degree of similarity between advertisements and requests, on the basis of the ontologies available to the Web services and the matching engine. Despite its flexibility the matching engine should maintain validity and recognize when an advertisement and a request describe functions that are so different that the matching process fails.

A capability matching algorithm for DAML-S profiles has been proposed in [12]. The basic idea underlying the matching algorithm is to verify whether the function described by the advertisement can be used in place of the function described by the request. More precisely, a match is recognized when the outputs of the request are subsumed by the outputs of the advertisement, so that the advertised function achieves all the results of the requester, furthermore, the inputs of the advertisement are subsumed by the inputs of the request, so that the requester has all the information that is needed to invoke the selected service. Degrees of matching are measured by the violations of the subsumption constraints, for example a match with a lower degree is recognized when the output of the advertisement is subsumed by the output of the request. Matching flexibility is constrained by existing DAML ontologies, no matching is recognized when no subsumption relation between advertisement and request is recognized.

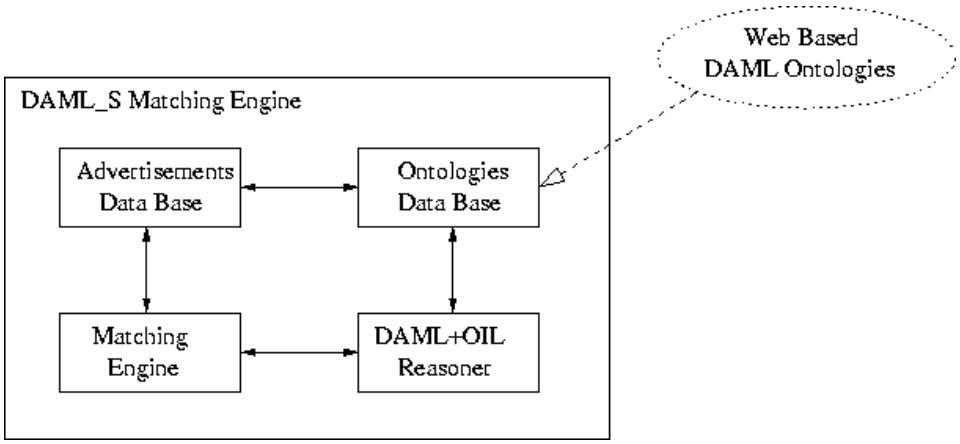

Figure 2: The architecture of the DAML-S Matching Engine

The matching algorithm has been implemented to control the *DAML-S Matchmaker* whose architecture is displayed in figure 2. Advertisements are stored in the *AdvertisementDB* component and indexed using ontologies downloaded from the Web and stored in the *OntologyDB.*

Upon receiving a request, the *Matching Engine* component selects advertisements that are relevant for the current request; then it uses the *DAML+OIL Reasoner* and the ontologies downloaded to select the advertisements that really match the request and compute the degree of such match.

Enabling capability matching in UDDI

The Universal Description Discovery and Integration (or UDDI) [15] is an Internet wide registry of Web services; because of its strong industry backing, UDDI is expected to become the standard registry for Web services. UDDI allows businesses to register their contact points, and a host of useful information about Web Services such as binding information to allow Web services to interact.

In addition, UDDI supports the association of an unbounded set of properties to the description of Web Services via a construct called *TModel*. TModels can be used to tag the type of service advertised and to provide abstract keys to be associated with a service specific value. For example, a service may specify its category using the North American Industry Classification System (hereafter NAICS) [16] published by the US Census. While TModels support the association of any type of data with the advertisement, their meaning is not codified, therefore two different

TModels may have the same meaning. Ultimately, their ability of describing a Web service is conditioned on a shared understanding of their meaning.

UDDI supports only a keyword based search of businesses, services and TModels in its repository. In addition services can be searched by their type specification through TModels. For instance, it is possible to search for all the services that adhere to the WSDL representation or that have a specific value associated with a TModel. Since search in UDDI is restricted to keyword matching, no form of inference or flexible match between keywords can be performed.

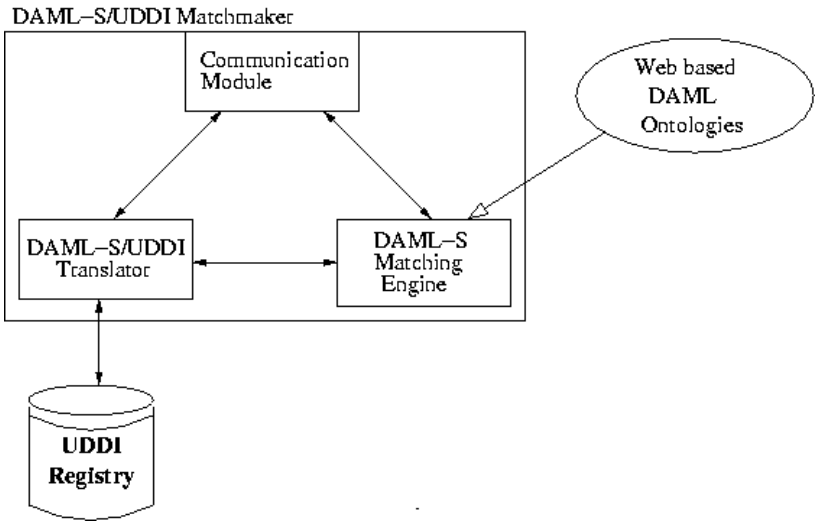

Figure3:The architecture of the DAML-S/UDDI Matchmaker

The limitation of UDDI is its lacking of an explicit representation of the capabilities of the Web Service. The result is that UDDI supports the location of essential information about the Web service, once it is known that the Web service exists, but it is impossible to locate a Web service only on the bases of what problems it solves. To solve this problem, a translation function from DAML-S Profiles to UDDI record has been proposed [13]. At its core of this function defines a set of TModels that correspond to properties of DAML-S Profiles therefore allowing any DAML-S profile to be recorded as UDDI record[1].

The DAML-S/UDDI Matchmaker[2] uses the DAML-S translation function described above to map DAML-S advertisements into UDDI records, and then it uses the UDDI registry to store and retrieve them. Furthermore, leveraging on DAML-S capability representation, the DAML-S/UDDI Matchmaker adds a semantic layer that performs a based capability matching between advertisements and requests using the matching engine described above and DAML ontologies published on the Web. The result of this work is empowering UDDI with DAML-S capability representation and with the corresponding matching mechanism to select Web services on the bases of their capabilities.

The architecture of the combined DAML-S/UDDI Matchmaker is described in figure 3. The Matchmaker receives advertisements and requests from outside through the *Communication Module*; upon recognizing that a message is an advertisement, the Communication Module sends it to the *DAML-S/UDDI Translator* that constructs a UDDI service description using information about the service provider, and the service name. The result of the registration with UDDI is a reference ID of the service. This ID combined with the capability description of the advertisement are sent to the DAML-S Matching Engine that stores the advertisement for capability matching. Requests follow the opposite direction: the Communicator Module sends them to the DAML-S Matchmaker that performs the capability matching. The result of the matching is the advertisement of the providers selected and a reference to the UDDI service record. The combination of UDDI records and advertisements is then send to the requester.

This system show the limits of UDDI and the value added by DAML-S and its support for functional descriptions and matching upon functional descriptions of services. Current work on this project attempts to reach a closer integration between UDDI and DAML-S by modifying the UDDI API to allow direct queries for capabilities and a complete integration of the DAML-S Matching engine with UDDI.

Managing Web services Interaction

The location of providers is an essential step toward a business interaction, but the conclusion of the business requires the direct negotiation between the buyer and the seller. E-commerce transactions are more complicated than traditional client/server transactions in which the client just asks questions to the server, rather e-commerce transactions

[1] Note that mapping from UDDI to DAML-S Profiles is meaningless, since some of the information stored in UDDI is not represented in the Profile, but in other modules such as the DAML-S Grounding.

[2] The DAML-S Matchmaker is available at www.damlsmm.ri.cmu.edu.

proceed through a refinement process in which the buyer and the seller respond to each other until they agree on the product and the price. For example, a transaction with an on-line travel agent typically requires the user to specify the date of travel, departing and arrival locations, but then the Web site proposes different itineraries and asks the buyer to make a selection. Still, e-commerce transaction are very structured so that both parties always know whose turn is next, and what kind of information to expect. This is in contrast with unrestricted conversations that can be generated by software agents which are governed only by the specification of speech acts that can be combined in any arbitrary way [9].

The DAML-S Process Model assumes that Web services will interact in a way that will be more similar to the interaction with Web sites than inter-agent communication. This view is consistent with growing infrastructure for Web services and proposed interaction specifications such as BPEL4WS [4]. 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.

Operationally, a Process Model is defined as an ordered collection of processes organized on the basis of a workflow which specifies the sequence of processes performed by the provider during the transaction. Each process is defined as a transformation between an initial state and a final state, where the initial state is specified by the inputs of the process and the preconditions for the process to run successfully. The final state 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 between external and internal input and outputs: external input and outputs correspond to incoming and outgoing messages between the provider and the requester; internal input and outputs feed into each other within the workflow of the provider and are used to specify what the provider does with the information received from the requester.

During the interaction with the provider, the requester analyzes the process model to infer the messages that they will exchange, specifically, the requester attempts to infer the inputs the provider needs and the outputs that result from the execution of the process. Ultimately, by following the process model of the provider and interpreting the information received, 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 DAML-S Grounding transforms the abstract description of the information exchanges between the provider and the requester into messages that can be exchanged on the net, 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.

A Web Service Architecture

DAML-S Process Model and Grounding provide the specifications of the interaction between the provider and the requester, but they leave open the problem of making use of the information that they provide in real implemented Web Services. In this paper, we propose a Web Service architecture that uses the information available in DAML-S descriptions, and specifically the Process Model and the Grounding, to interact with other Web services

The architecture we propose is described in the diagram in figure 4. The diagram can be divided in three areas: on the right is the *Application* which represents the main body of the Web service. Applications may range from financial consulting to travel booking to news reporting. Crucially, DAML-S does not make any assumption on the application, nor on its internal structure therefore, from our point of view the application is just a black box.

The application communicates with other Web services through the *DAML-S port* that is defined in the middle layer. The DAML-S port is composed of three modules that are activated in sequence. The first module is a *DAML parser* that transforms DAML-S specifications for the Process Model and the Grounding into a set of predicates that provide a representation equivalent to the DAML-S descriptions, but in a format that can be interpreted by the JESS theorem prover [10]. The predicates are then passed to the *DAML-S Processor* that uses the Process Model and Grounding specifications to extract the interaction protocol with the provider and compose the abstract messages to exchange with the provider. Furthermore, the DAML-S Processor interacts with the Application to extract the information that forms the content of outgoing messages or to provide the information received through incoming messages. Such a mapping is used by the *Web service Invocation* module to contact other Web services. Messages received by Web services follow the opposite path, they are first parsed by the Web service Invocation module that

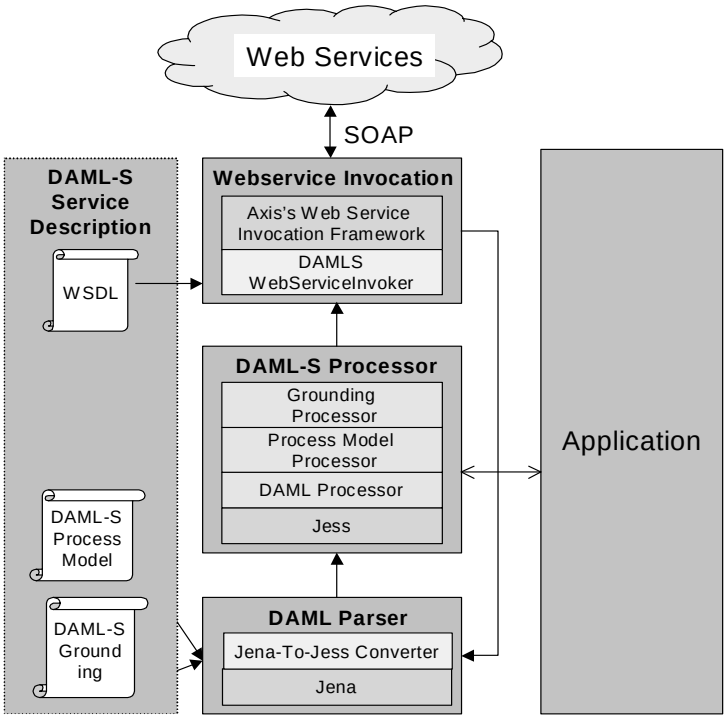

Figure 4: Description of DAML-S Web Service architecture

extracts the content of the messages, then they are parsed and transformed into a Jess predicates and finally analyzed by the DAML-S Processor.

The diagram also shows the information used by the DAML-S port and the modules that access it. Specifically, the Process Model and Grounding are passed to the Parser, while the WSDL module is passed directly to the Invocation module that

The DAML-S processor is the core of our architecture, it is responsible for transforming the DAML-S specifications into abstract messages to be send to other Web services. Furthermore, the DAML-S Processor is responsible for the interaction with the rest of the application. The DAML-S processor is constructed as a stack of processors. At the bottom of the stack lies Jess [10], a Java implementation of CLIPS [3]. Jess performs a forward and backward inferences to derive the consequences of the knowledge it receives. A *DAML Processor* is built on top of Jess, using the DAML Jess KB framework [6], to provides an implementation of the DAML language within Jess. The result of the combination of DAML Jess KB and Jess is a DAML inference engine that we use to derive inferences from the ontologies loaded as well as the Process Model and the Grounding. The *Process Model Processor* and *Grounding Processor* contain rules that describe the semantics of DAML-S Process Model and Grounding. Specifically, they contain rules on the location of the next process to execute, its inputs and outputs as well as their transformations in abstract messages.

In the previous sections we noted that DAML-S is mute with about to the Web service application and concentrates on the specifications of the interaction between Web services. Nevertheless, the application level is responsible for many of the 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. As a consequence, the neat picture shown in figure 4 results only partially true in real applications where the DAML reasoning cannot be restricted to the management of the port,

but should be extended to the whole Application. In ultimate analysis, DAML-S requires applications that look more like intelligent software agents than traditional e-commerce applications.

Test Application

To test our approach to DAML-S we implemented a B2C application in the travel domain in which Web services coordinate with other applications like MS Outlook to organize a trip to a conference: the DAML PI Meeting. The architecture of the application is shown in figure 5.

The scenario of the demonstration is the following, the user upon learning of the PI Meeting, loads the schedule of the conference annotated as a DAML ontology from the conference Web site using the Retsina Calendar Agent (RCal) [14], and asks RCal to plan a trip to the conference. RCal infers the schedule of the conference from the DAML annotated schedule and then uses that information to contact the DAML_S Matchmaker described above to book locate Web Services for flight reservation and the Car Rental.

Upon locating and selecting the Web services, the RCal agent interact with them using the DAML-S port described above, which results in a booking of the flight to the conference and the car rental.

Finally the RCal records the schedule of the trip in the MS Outlook calendar of the user.

In the implementation of the application, care was taken to reduce the hardcoding of communication information in the Calendar Agent, rather communication details have been inferred by the DAML-S Port described above using only Process Model and Grounding specifications. In addition we had to employ the HiTAP planner [11] to plan the trip to the conference. HiTAP is based on HTN planning [Erol94], but it also allows for interleaving of planning and execution which is fundamental to control the interaction between Web services since at planning time the Web service does not know which services are available, whether they will provide the information desired or what information will they need.

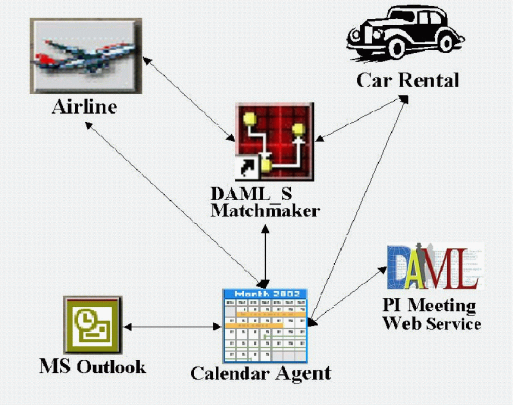

Figure 5: The Application Architecture

Conclusions and Further Work

In this paper we described how to use DAML-S to control the interaction between Semantic Web services. Specifically, we used DAML-S to describe capabilities of Web services so that they can find each other on the basis of the information that they provide, rather than incidental properties such as their name, port, or a free text description of what they do. Furthermore, we showed how DAML-S can also be used to control the autonomous interaction between Web services without any need of pre-programming hardcoding neither the sequence of messages to exchange nor the information to be transmitted.

The work presented here shows the importance of the Semantic Web and the need for widespread ontologies, to the point that without ontologies, this work would not be possible. In the Web service discovery phase, ontologies provide the basic information needed to describe input and outputs of Web services, as well as preconditions and effects. For one, inputs, outputs, preconditions and effects need to refer to objects or concepts in the World for which all the parties in the transaction need to have a shared knowledge and understanding. Furthermore, ontologies provide an inference framework that allows Web services to resolve discrepancies and mismatches between the knowledge that they are using. This is particularly relevant in the DAML-S/UDDI matchmaker that has to abstract from the superficial differences between the advertisement and the request to recognize whether they describe the same function. Finally, ontologies play an essential role during Web services interaction, because they provide a shared dictionary so that Web services can understand each other messages. In addition, ontologies provide the basics for the use of the knowledge exchanged by Web services by supporting inferences when new knowledge is added.

Our work highlights some of DAML-S advantages. The application described in this paper shows that indeed DAML-S supports autonomous invocation of Web services; but also shows that DAML-S provides only the base for autonomous invocations and that it poses hard requirements on the side of the application that have to make very difficult decisions to take advantage of DAML-S. It is still an open question whether DAML-S provides any advantage over other technologies even in absence of such powerful decision making mechanisms. Indeed, it could be claimed that the use of ontologies provide a more flexible interaction mechanism compared with the use of pure XML data type

as supported by XML Schemas because mismatches between the information received and the information expected may be resolved through logical inference.

Future work toward the completion of a DAML-S toolkit that supports the implementation of Web services involves a tighter integration between the DAML-S port and the discovery mechanism. Specifically, we need to address issues like automatic generation of advertisements from the problem description and criteria for the selection of the best provider out of a pool of candidates. Furthermore, the Matchmaker may provide the rational of behind the matches that it found. Such matching rational may be of use both during the selection process, and during the interaction providing key information to be used to interpret messages exchanged by the Web services.

Bibliography

[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] *CLIPS*: http://www.ghg.net/clips/CLIPS.html

[4] F. Curbera, Y. Goland, J. Klein, Microsoft, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana: *Business Process Execution Language for Web Services, Version 1.0*: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[5] DAML Joint Committee.: *Daml+oil (march 2001) language*.: http://www.daml.org/2001/03/daml+oil-index.html 2001

[6] *DAML Jess KB*: http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/

[7] The DAML-S Coalition.: *Daml-s: Web service description for the semantic web*: In ISWC2002.

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

[9] The Foundation for Physical Agents (FIPA): *FIPA ACL*: http://www.fipa.org

[10] Jess: http://herzberg.ca.sandia.gov/jess/

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

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

[13] 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.

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

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

[16] United States Census Bureau.:*North american industry classification system (NAICS)*.: http://www.census.gov/epcd/www/naics.html 1997.

[17] W3C.:*Soap version 1.2, w3c working draft 17 december 2001*.: http://www.w3.org/TR/2001/WD-soap12-part0-20011217/ 2001.