Semantic e-Workflow Composition

Jorge Cardoso and Amit Sheth
LSDIS Lab, Department of Computer Science
University of Georgia
Athens, GA 30602 – USA
jcardoso@uga.edu, amit@cs.uga.edu

Abstract

Systems and infrastructures are currently being developed to support Web services. The main idea is to encapsulate an organization's functionality within an appropriate interface and advertise it as Web services. While in some cases Web services may be utilized in an isolated form, it is normal to expect Web services to be integrated as part of workflow processes. The composition of workflow processes that model e-service applications differs from the design of traditional workflows, in terms of the number of tasks (Web services) available to the composition process, in their heterogeneity, and in their autonomy. Therefore, two problems need to be solved: how to efficiently discover Web services – based on functional and operational requirements – and how to facilitate the interoperability of heterogeneous Web services. In this paper, we present a solution within the context of the emerging Semantic Web, that includes use of ontologies to overcome some of the problems. We start by illustrating the steps involved in the composition of a workflow. Two of these steps are the discovery of Web services and their posterior integration into a workflow. To assist designers with those two steps, we have devised an algorithm to simultaneously discover Web services and resolve heterogeneity among their interfaces and the workflow host. Finally, we describe a prototype that has been implemented to illustrate how discovery and interoperability functions are achieved.

Keywords: Web Services Composition, e-Workflows, Web Services discovery, Web Services interoperability, Semantic Web, Ontology-based systems, Semantic Heterogeneity, Workflow QoS

1 Introduction

E-services have been are announced as the next wave of Internet-based business applications that will dramatically change the use of the Internet (Fabio Casati, Ming-Chien Shan *et al.* 2001). With the development and maturity of infrastructures and

solutions that support e-services, we expect organizations to incorporate Web services as part of their business processes. While in some cases Web services may be utilized in an isolated form, it is natural to expect that Web services will be integrated as part of workflows (Berners-Lee 2001; Fensel and Bussler 2002). Workflow management systems are capable of integrating business objects for setting up e-services in an amazingly short time and with impressively little cost (Shegalov, Gillmann *et al.* 2001). Workflows and Web services play a major role in architectures such as business-to-business (B2B), business-to-customer (B2C), customer-to-customer (C2C), dynamic trading processes (Sheth, Aalst *et al.* 1999), dynamic value chains (Lee and Whang 2001), virtual organizations, and virtual Web organizations (Ulrich 2001).

A workflow is an abstraction of a business process. It comprises a number of logic steps (known as tasks or activities), dependencies among tasks, routing rules, and participants. In a workflow, a task can represent a human activity or a software system. The emergent need of workflows to model e-service applications makes it essential that workflow tasks be associated with Web services. As a result, research is currently being carried out to enhance workflows systems in their support and management of Web services (Shegalov, Gillmann *et al.* 2001).

The modeling of e-services using workflows raises two challenges for workflow systems. First, Web services must be located that might contain (a) the desired functionality and (b) operational requirements needed to carry out the realization of a given task. It is necessary to efficiently discover Web services from the potentially thousands of services available on the Internet. Second, once the desired Web services have been found, mechanisms are needed to (c) facilitate the resolution of structural and semantic differences. This is because the heterogeneous Web services found in the first step need to interoperate with other components present in a workflow host.

(a) The design of traditional workflow applications involves the selection of appropriate tasks with their desired functionality in order to compose a workflow and to establish connections among these tasks (control and data flow). Tasks are selected from a workflow repository (Arpinar, Miller *et al.* 2001; Song 2001) which typically contains only tens to a few hundreds of tasks. Since the number of tasks to choose from is modest, the process is humanly manageable, not requiring sophisticated search or discovery mechanisms. However, when a workflow is employed to model e-services, the potential number of Web services available for the composition process can be extremely large. Then, we are no longer searching for a task from a set of a few hundred, but we are searching for a service from a set that can potentially contain thousands of Web services. One cannot expect a designer to manually browse through all of the Web services available and select the most suitable ones.

(b) The autonomy of Web services does not allow for users to identify their operational metrics at design time, *i.e.*, before their actual execution. Operational metrics characterize Web services according to their Quality of Service (QoS), which includes their timeliness, quality of products delivered, cost of service, and reliability. When composing a workflow it is indispensable to analyze and compute its overall QoS (Cardoso, Miller *et al.* 2002; Cardoso, Sheth *et al.* 2002; Miller, Cardoso *et al.* 2002). This allows organizations to translate their vision into their business processes more efficiently, since workflows can be designed according to QoS metrics. The management of QoS directly impacts the success of organizations participating in electronic activities.

A good management of quality leads to the creation of quality products and services, which in turn fulfills customer expectations and achieves customer satisfaction. To achieve these objectives, one of the first steps is to develop an adequate QoS model for workflow processes, tasks, and Web services. Such a model will allow for the discovery of Web services and for the composition of workflows based on operational requirements.

(c) Numerous of the information interoperability problems that the composition of workflows involving Web services face are already well known within the distributed database systems community (Sheth and Larson 1990; Kashyap and Sheth 1996; Calvanese, Giacomo *et al.* 1998; Parent and Spaccapietra 1998). To achieve interoperability, it is necessary to address the problem of semantic integration – the identification of semantically similar objects that belong to different systems and the resolution of their schematic differences (Kashyap and Sheth 1996). When tasks and Web services are put together, their interfaces (inputs and outputs) need to interoperate; therefore, structural and semantic heterogeneity needs to be resolved. Structural heterogeneity exists because Web services use different data structures and class hierarchies to define the parameters of their interfaces. Furthermore, semantic heterogeneity considers the intended meaning of the terms employed in labeling input and output parameters. The data that is interchanged among Web services has to be understood. Semantic conflicts occur when a Web service output connected to another service or task input does not use the same interpretation of the information being transferred. The general approach to semantic integration has been to map the local terms onto a shared ontology. Even though a shared ontology ensures total integration, constructing such an ontology is costly, if not impractical; autonomous systems are required to commit to a shared ontology, and compromises are difficult to maintain when new concepts are added (Rodríguez and Egenhofer 2002).

The main motivation for our work is the need to enhance workflow systems with better mechanisms for e-service composition. More precisely, we target the development of new mechanisms for Web services discovery and integration. Our method is novel and provides a multidimensional approach to Web service discovery and integration using syntactic, semantic, and operational metrics of Web services.

In this paper, we describe the composition process of e-workflows and present an algorithm to be employed when designers need to add Web services to an e-workflow. E-services can be orchestrated with hard-coded applications or by using workflows. We call a workflow which manages e-services and possibly traditional workflow tasks an e-workflow. Our approach relies on the use of ontologies to describe workflow tasks and Web services interfaces. Ontologies-based approaches have been suggested as a solution for information integration that achieves interoperability (Kashyap and Sheth 1994; Uschold and Gruninger 1996).

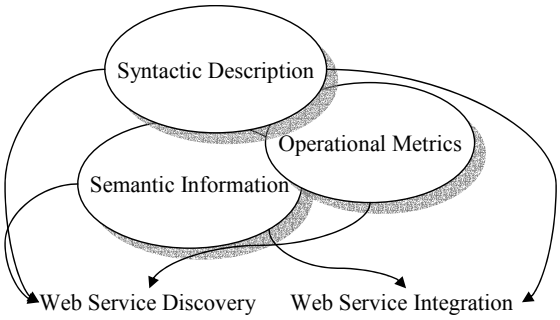

Figure 1-1 – Multidimensional approach to Web Service Discovery and Integration

The discovery and integration of Web services into e-workflows has specific requirements and challenges as compared to previous work on information retrieval systems and information integration systems. In this paper, we describe a methodology with the aim to give a solution to the following objectives and issues:

- Increase the precision of the discovery process. The search has to be based, not only on syntactic information, but also on Web services operational metrics and semantics.
- Tasks and Web services operational metrics need to be represented using a suitable model describing the QoS metrics of (Cardoso, Sheth *et al.* 2002).
- Enable the automatic determination of the degree of integration of the discovered Web services and a workflow host.
- The integration of Web services differs from previous work on schema integration due to the polarity of the schema that must be integrated. The polarity of schema forces an output schema to be connected to an input schema. Furthermore, an input schema needs to have all its input parameters satisfied. When a task or Web service is added to an e-workflow, it is necessary to integrate its input and output schema with the other tasks already present in the process. The input schema (ns_i) of a new task needs to be integrated with one or more output schema ($s_{o,r}$) of the tasks connected to it ($\{s_{o,1}, s_{o,2}, \ldots, s_{o,n}\} \rightarrow ns_i$). The output schema ($ns_o$) of the new task needs to be integrated with one or more input schema ($s_{i,r}$) of the tasks it connects to ($ns_o \rightarrow \{s_{i,1}, s_{i,2}, \ldots, s_{i,n}\}$). This process does not require a full integration of the schema $\{s_{o,1}, s_{o,2}, \ldots, s_{o,n}\}$ with the schema ns_i. Only the input schema ns_i needs to have its schema fully integrated, *i.e.* in order to work properly all its (mandatory) inputs need to be mapped to an output belonging to one of the schema $s_{o,r}$. For the integration of the output schema s_o, the schema $\{s_{i,1}, s_{i,2}, \ldots, s_{i,n}\}$ are the ones that need to be fully integrated.
- Previous work (Paolucci, Kawamura *et al.* 2002) on Web service discovery does not address the interoperability problem. Furthermore, the algorithm developed does not address the problem of matching outputs/inputs defined in distinct ontologies. This is a strong limitation. Since Web services are heterogeneous, autonomous, and developed independently, it is desirable to compare and discover Web services that have their schema defined by different ontologies.

This paper is structured as follows. Section 2 presents a scenario illustrating the composition of an e-workflow and highlights the difficulties involved. Section 3 focuses on the extension of traditional workflow tasks specifications to semantically describe their interfaces, on the specification of Web services, and on the association of a QoS model to specify operational metrics for both tasks and Web services. In section 4, we describe the composition process of an e-workflow and the structures that are created and manipulated; these will later be used in the Web service discovery phase. Section 5 represents the core of our work; we present an algorithm that takes into account syntactic, operational, and semantic information in order to compute the degree of similarity of a Web service template and a Web service object. The algorithm evaluates the similarity of its arguments based on their degree of integration. Section 6 presents the architecture of the prototype we have developed to demonstrate the concepts introduced in this paper. Section 7 discusses related work, and section 8 presents our conclusions.

2 Scenario

A designer is composing an e-workflow to automatically manage the approval of travel authorization requests to conferences. A partial view of the workflow design is illustrated in Figure 2-1. Another interesting example, which could be cast to the e-workflow composition process, is described in (Barbar, Mehrothra *et al.* 1996). The workflow manages the arrangement, cancellation, and postponement of office meetings.

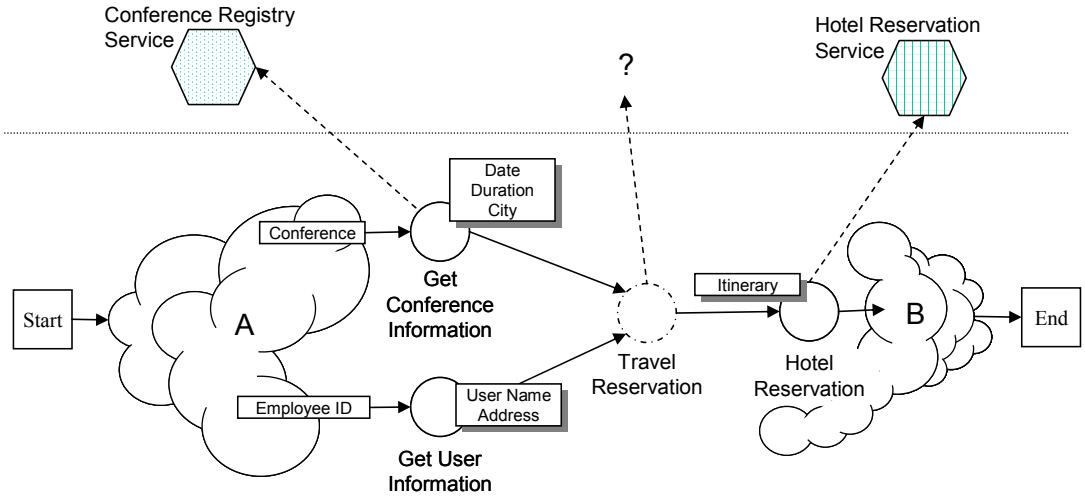

Figure 2-1 – Travel Authorization Request e-Workflow

The e-workflow operates in the following way. When an employee desires to attend a conference, he initializes an instance of the travel authorization request e-workflow. The first part of the e-workflow is the approval process; it is represented by the letter 'A' in the figure. The approval process allows managers to decide if an employee's request will be approved (we have hidden this portion of the workflow for brevity to reduce its complexity.)

If the managers approve the request, the next tasks to be executed are *Get Conference Information*, *Get User Information*, *Travel Reservation*, and *Hotel*

Reservation. The *Get Conference Information* task is responsible for obtaining the date, duration, and the city where the conference is being held, based on the conference name. To obtain this information a Web service is chosen and linked to a workflow task. The *Get User Information* task retrieves the employee's name and address based on his ID. The *Travel Reservation* task is responsible for making a travel reservation according to the conference date, duration, city; it is also based on the employee's personal information. Finally, the *Hotel Reservation* task makes the necessary hotel reservation based on the travel itinerary.

Once the tasks involved with the travel and hotel reservation are executed, the portion of the e-workflow represented by the letter 'B' is executed. This part of the e-workflow is responsible for notifying the user of the travel arrangements made for him.

Let us assume that the designer has already placed the tasks shown in Figure 2-1 on the canvas. The e-workflow is almost complete; only the *Travel Reservation* task realization is missing. The designer manually looks for an appropriate Web service by browsing the Internet. This process is time consuming, cumbersome, and tedious. Potentially tens or hundreds of thousands of on-line Web services may be available. Only hundreds provide the desired functionality, and maybe only a handful provide the required operational metrics and interface (*i.e.*, input and output parameters). Furthermore, once a suitable Web service has been found, it needs to be integrated with the tasks already placed in the workflow. The designer needs to manually establish data connections among the new Web service and the tasks already present in the e-workflow, accounting for structural and semantic differences.

2.1 E-Workflow Composition Problems

In the previous scenario, the workflow designer faces two problems: locating a Web service with the desired functionality and operational metrics to accomplish a specific task and resolving the structural and semantic differences between the service found and the tasks and Web services to which it will be connected (using transitions).

We cannot expect a designer to discover a Web service manually, since potentially thousands of services are available on the Internet. Thus, efficient discovery mechanisms must be available. What makes the e-service vision attractive is the ability to automatically discover the e-services that fulfill users' needs (Fabio Casati, Ming-Chien Shan *et al.* 2001). The discovery of a Web service cannot only be based on its name or description; it also has to account for its operational metrics and its interfaces.

The composition of e-workflows cannot be undertaken while ignoring the importance of operational metrics. Trading agreements between suppliers and customers modeled with e-workflow include the specification of QoS items such as products or services to be delivered, deadlines, quality of products, and cost of service. The correct management of such specifications directly impacts the success of organizations participating in e-commerce and also directly impacts the success and evolution of e-services itself.

Web services can be seen as black boxes, with an input interface and an output interface. Since, when integrated into an e-workflow, a Web service has to interoperate at the interface level with adjacent tasks, the discovery also has to be based on the structural and semantic properties of its inputs and outputs. Once a Web service is found, it is not

realistic to expect that its interfaces will perfectly match and interoperate with the hosting e-workflow without additional work. Web services are heterogeneous by nature; we expect the designer will need to manually establish connections among the Web service interfaces and the tasks present in an e-workflow. In our example, the designer is faced with the problems of manually connecting the outputs of the tasks *Get Conference Information* and *Get User Information* with inputs of the task *Travel Reservation*, and then connecting the outputs of the task *Travel Reservation* with the inputs of the task *Hotel Reservation*. To facilitate this work, a workflow designer should be assisted by mechanisms that suggest the establishment of a connection between outputs and inputs that maximizes the degree of integration.

3 Workflow Tasks and Web Service Tasks

We rely on the use of ontologies to semantically describe task and Web service interfaces. Semantics have been a strong candidate for increasing the success of information discovery and integration on the Internet; its use has been presented as the next step in the evolution of the World Wide Web (Berners-Lee and Fischetti 1999; Fensel and Musen 2001).

The importance of ontologies is being recognized in research fields as diverse as knowledge engineering, knowledge representation, qualitative modeling, language engineering, database design, information modeling, information integration, object-oriented analysis, information retrieval and extraction, knowledge management and organization, and agent-based systems design (Guarino 1998). Ontologies are introduced as an "explicit specification of a conceptualization" (Gruber 1993). The use of ontologies for the explication of knowledge is a possible approach to overcome the problem of integrating heterogeneous workflow tasks and Web services. In nearly all ontology-based integration approaches, ontologies are used for the explicit description of the information source semantics. Therefore, they can be used to describe the semantics of task interfaces, making their content and function explicit and thus enhancing the integration process.

3.1 Ontologies

An ontology $\Omega_i = \{c_1, \ldots, c_n\}$ contains a set of classes. Each class c_j has an associated set of properties $P_k = \{p_1, \ldots, p_m\}$. Each property has a range indicating a restriction on the values the property can take. An ontology relates more specific concepts to more general ones (from which generic information can be inherited). Such links have been variously named "is a," "subset of," "member of," "subconcept of," "superconcept," *etc*. Such links are used to organize concepts into a hierarchy or some other partial ordering, called a "taxonomy." The taxonomy is used for storing information at appropriate levels of generality and automatically making it available to more specific concepts by means of a mechanism of inheritance. More general concepts in such a partial order are said to subsume more specific concepts, and a more specific concept is said to inherit information from its subsumers. The notion of ontological concepts is very similar to the notion of classes in object-oriented programming.

In our implementation, tasks and Web services interfaces are semantically described by concepts (classes) that are defined in ontologies constructed with DAML+OIL

(Horrocks, Harmelen *et al.* 2001). Our approach is not dependent on DAML+OIL; other ontology representation languages could be employed. DAML+OIL enables the creation of ontologies for any domain, and it is a particularly suitable framework that makes the description of services computer-interpretable and shared.

3.2 Extending Workflow Tasks Specifications

In most workflow systems, each task is described by several elements which typically include a name, a type, a list of input parameters and output parameters, a short textual description, and a task realization (implementation). A task invocation specifies the number of input parameters that must be supplied for a proper task realization and the number of outputs parameters to hold and transfer the results of the task realization to other tasks. In their simplest form, the input and output parameters can be represented by attributes, or they can follow an object-oriented model represented by data components. Attributes are specified with an attribute name, a type, and an optional initial value. Examples of built-in primitive types include Boolean, string, byte, integer, and real. Data components are represented by classes composed of a collection of attributes. Classes may form a hierarchy in which inheritance is allowed.

To enhance the integration of tasks and Web services, workflow components need to have their inputs and outputs associated with ontological concepts (classes). This will facilitate the resolution of structural and semantic heterogeneity. Since there is a strong analogy between the attributes and data classes of an object-oriented model and the concepts classes defined in an ontology, the establishment of mappings between the two is facilitated. Figure 3-1 illustrates the establishment of such a mapping.

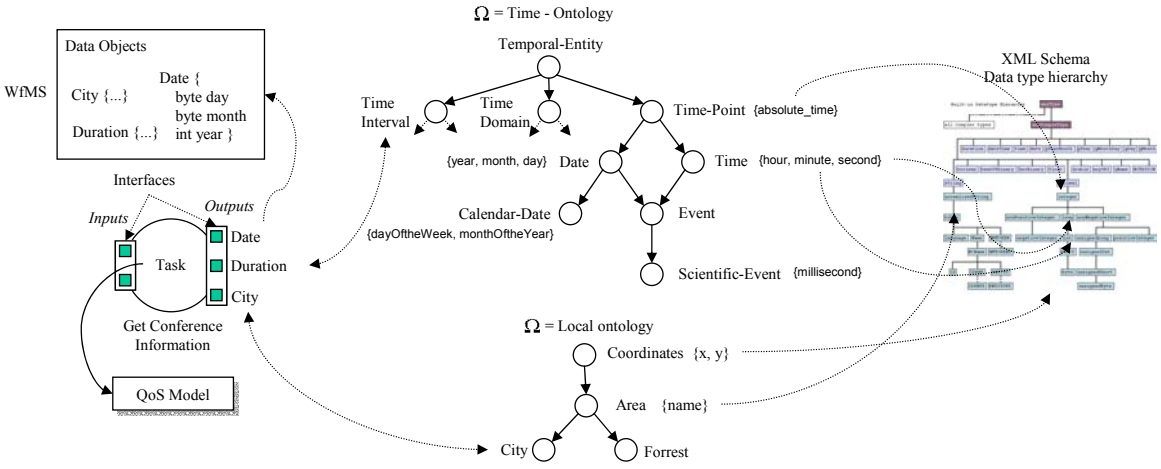

Figure 3-1 – Association of task inputs and outputs with concepts

Each input and output data class parameter of a task is associated with an ontological concept class. We assume a one-to-one mapping between a data class and its associated concept class; *i.e.* each attribute of a data class must have a corresponding property that belongs to the associated concept class. This assumption can be further relaxed by considering work in schematic heterogeneity (Kashyap and Sheth 1996) and schema mapping (Madhavan, Bernstein *et al.* 2001).

Primitive data types of attributes (such as byte and double) are represented in the ontology by properties which reference data types defined in the XML Schema specification (XMLSchema 2001). It would have been possible to associate primitive built-in data types with ontological concepts or properties. Nevertheless, we have chosen XML Schema because it provides a comprehensive data type hierarchy, which includes unsigned byte, short, decimal, non-negative integer, string, and base 64 binary.

3.3 Web Service Specification

The emergence and challenges of e-services have directed the development and creation of mechanisms to support Web services. One fundamental issue is their specification. Two main approaches have been proposed. One of the approaches uses declarative and structured data based purely on syntax, such as WSDL (Christensen, Curbera *et al.* 2001) and XLANG (Thatte 2001). A second approach provides a semantic orientation to the description of Web services. This is the case in the DAML-S specification (Ankolekar, Burstein *et al.* 2001).

Web services are "self-contained, self-describing modular applications that can be published, located, and invoked across the Web" (Tidwell 2000) and therefore are a modern alternative to the specification of workflow tasks. Since they are self-described, the interoperation among independently developed Web services is facilitated. Traditional workflow tasks, such as non-transactional, transactional, and human tasks (Kochut, Sheth *et al.* 1999) can easily be represented or encapsulated with Web services.

As with WSMF (Fensel and Bussler 2002), our approach to e-workflow composition is not dependent on the method chosen to specify Web services. Therefore, any of the specification languages mentioned above can be employed. For the prototype that we have developed we have selected the DAML-S specification; more precisely, we use the *Service Profile* ontology.

The service profile ontology describes the functionality of a Web service. It tells "what the service does" (Ankolekar, Burstein *et al.* 2001) and is employed to advertise Web services availability and capability. We have decided to use DAML-S because in the same way we did with workflow tasks, we need to establish associations among the inputs and outputs parameters of a Web service with ontological concepts. Since the DAML-S specification semantically describes Web services, there is an explicit association of Web services interface with concepts. In Listing 3-1 we give a partial example of the specification of a Web service using DAML-S.

One of the service inputs is the *PreferredClass*, and one of the outputs is the *TripItinerary*. Both of them refer to concepts defined in the ontology *itinerary-ont.daml*.

When using a declarative specification language such as WSDL, there is also the need to associate each input and output with an ontological concept so that they can be semantically described. This may require the extension of the Web service specification language to include additional tags which will be employed to specify the ontology and the concepts associated with input and output parameters.

```
- <profile:input>
- <profile:ParameterDescription rdf:ID="PreferredClass">
  <profile:parameterName>PreferredClass</profile:parameterName>
  <profile:restrictedTo rdf:resource="http://
www.daml.org/2001/06/itinerary/itinerary-ont.daml#class" />
   </profile:ParameterDescription>
  </profile:input>

- <profile:output>
- <profile:ParameterDescription rdf:ID="Itinerary ">
  <profile:parameterName>TripItinerary</profile:parameterName>
  <profile:restrictedTo rdf:resource="
http://www.daml.org/2001/06/itinerary/itinerary-ont.daml#Flight" />
    </profile:ParameterDescription>
  </profile:output>
```

Listing 3-1 – Web service specification using DAML-S

3.4 Operational Metrics

The operational metrics of tasks and Web services are described using a QoS model. For us, QoS represents the quantitative and qualitative characteristics of an e-workflow application which are necessary to achieve a set of initial requirements. E-workflow QoS addresses the operational issues of workflows, rather than workflow process functions. Quantitative characteristics can be evaluated in terms of concrete measures such as workflow execution time, cost, reliability, etc. Qualitative characteristics specify the expected services offered by the system such as security and fault-tolerance mechanisms. QoS should be seen as an integral aspect of workflows, and therefore it should be integrated with tasks and Web services specifications.

While the DAML-S specification that we use includes constructs to specify quality of service parameters, such as quality guarantees, quality rating, and degree of quality, the specification does not provide a detailed set of classes and properties to represent quality of service metrics. The model needs to be extended to allow for a precise characterization of each dimension in order to permit the implementation of algorithms for the automatic computation of QoS metrics of processes based on their sub-processes' QoS metrics. Therefore, we have developed our own model.

We have investigated relevant work to determine which dimensions would be relevant to compose a more suitable QoS model for the automatic computation of QoS metrics.

Based on previous studies (Garvin 1988; Stalk and Hout 1990; Rommel 1995), as well as our experience in the workflow domain, we have constructed a model composed of the following dimensions: *time*, *cost*, *reliability*, and *fidelity* (Cardoso, Sheth *et al.* 2002). Since *fidelity* is subject to judgments and perceptions, we have decided to omit its specification and analysis in this paper. Nevertheless, a thorough study can be found in (Cardoso, Miller *et al.* 2002).

While in this paper we do not discuss the computation of QoS metrics, comprehensive solutions to the difficult problems encountered in synthesizing QoS for composite services are discussed in detail in Cardoso, Sheth *et al.* (2002). This paper

presents a stochastic workflow reduction algorithm and discusses the use of simulation analysis (Miller, Cardoso *et al.* 2002) for computing aggregate QoS properties step-by-step.

3.4.1 QoS Dimensions

Based on our model, we have we have developed an ontology for the specification of QoS metrics (for tasks and Web services). This information will allow for the discovery of Web services based on operational metrics and includes the following dimensions:

Time is a common and universal measure of performance. Task response time (T) corresponds to the time a workflow instance takes to be processed by a task. The *task response time* can be broken down into two major components: *delay time* and *process time*. Delay time (DT) refers to the non-value-add time needed in order for an instance to be processed by a task. Process time (PT) is the time a workflow instance spends at a task while being processed; in other words, it corresponds to the time a task needs to process an instance.

Cost (C) represents the cost associated with the execution of workflow tasks. During workflow design, prior to workflow instantiation, and during workflow execution it is necessary to estimate the cost of its execution to guarantee that financial plans are followed. Task cost is the cost incurred when a task or Web service is executed; it can be broken down into two major components: *enactment cost* and *task realization cost.* The enactment cost (EC) is the cost associated with the management of the workflow system and workflow instances monitoring. The task realization cost (RC) is the cost associated with the runtime execution of the task.

Task **Reliability** (R) corresponds to the likelihood that the components will perform when the user demands them. It is a function of the failure rate. Each task structure has an initial state, an execution state, and two distinct terminating states. One of the states indicates that a task has failed or was aborted, while the other state indicates that a task is done or committed (Krishnakumar and Sheth 1995). This QoS dimension provides information concerning the relationship between the number of times the state done/committed is reached, and the number of times the failed/aborted state is reached. To describe task reliability we follow a discrete-time modeling approach. Discrete-time models are adequate for systems that respond to occasional demands, such as database systems. We use the stable reliability model proposed by Nelson (1973), for which the reliability of a task t is $R(t) = 1$ - failure rate.

3.4.2 Dimensions Characterization

For each dimension, the description of the operational runtime behavior of a task is composed of two classes of information: *basic* and *distributional*.

The basic class associates with each task's QoS dimension the minimum value, average value, and maximum value the dimension can take. For example, the cost

dimension corresponds to the minimum, average, and maximum cost associated with the execution of a task.

The second class, the distributional class, corresponds to the specification of a constant or of a distribution function (such as Exponential, Normal, Weibull, or Uniform) which statistically describes task behavior at runtime. The values specified in the basic class are typically employed by mathematical methods in order to compute workflow QoS metrics, while the distributional class information is used by simulation systems to compute workflow QoS.

Table 3-1 shows an example of the specification of QoS metrics for a task from a genomic workflow (Cardoso, Miller *et al.* 2002).

	Basic class			Distributional class
	Min value	Avg value	Max value	Dist. Function
Time	192	196	199	Normal(196, 1)
Cost	576	576	576	576.0
Reliability	100%	100%	100%	1.0

Table 3-1 – Task QoS for a manual task

4 The e-Workflow Composition Process

The composition of e-workflows differs slightly from the design of traditional workflows. A typical scenario of the composition process is as follows. The designer composes an e-workflow for which several traditional workflow tasks (*e.g.* human, non-transactional, and transactional tasks) and Web service tasks have already been placed and interconnected on the canvas. Tasks with a realization are called grounded tasks (GT). When the designer wishes to add a Web service to the workflow, he starts by creating a service template (ST) – see section 4.1 for the formal specification of a ST – which indicates his intention to extend the functionality of the workflow. The ST will be employed later to find an appropriate Web service.

Once a ST is created, it is sent to the Web service discovery module, which returns a set of service object (SO) references that are ranked according to their degree of similarity with the service template. Services can be ranked according to a syntactical, operational, or semantic perspective. The designer then selects the most appropriate Web service to accomplish his objectives (section 6 shows an example of the SOs retrieved from the discovery process). The selection automatically associates a realization with the ST, causing it to change its state to a grounded task. Additionally, a set of data mapping is presented to the designer suggesting a possible interconnection among the newly created task interfaces and the grounded task interfaces.

A ST has five sections that need to be specified:

- The name of the Web service to be found,
- Its textual description,

- Its operational metrics,
- The set of outputs parameters from the grounded tasks that will be connected to SO inputs, and
- The set of input parameters from the grounded tasks that a SO will be connected to.

The construction of a ST is illustrated in Figure 4-1. The outputs of the GTs *Get Conference Information* and *Get User Information* (*Date*, *Duration*, *City*, *User Name*, and *Address*) are employed to construct the outputs of the ST. The input of the GT *Hotel Reservation* (*Itinerary*) is employed to construct the inputs of the ST. The user manually sets the name, description, and QoS model of the Web service to be found.

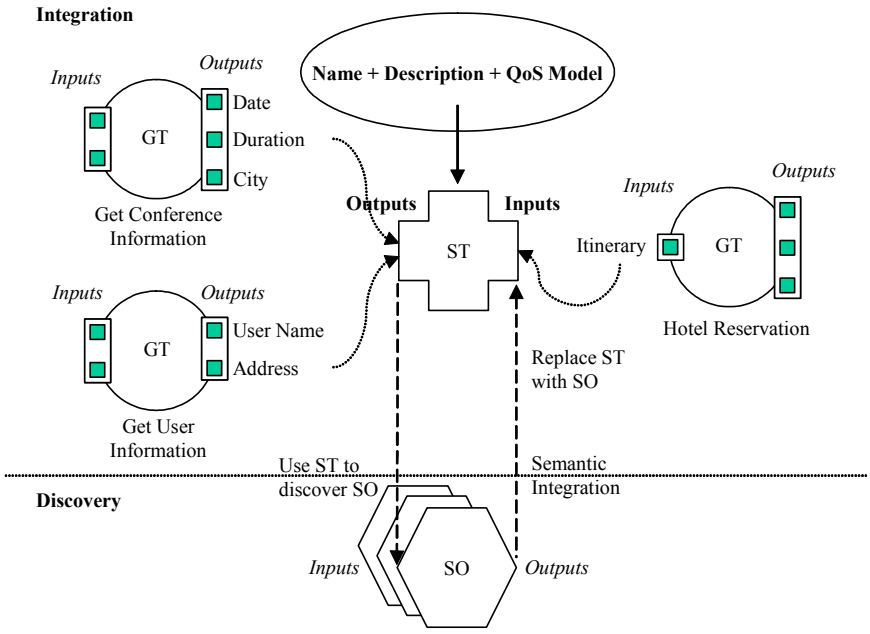

Figure 4-1 – GT, ST, and SO structures

4.1 E-Workflow Integration Components

The composition process described in the previous section involved the manipulation of three distinct structures: GT, ST, and SOs. In this section, we formally describe each structure.

Grounded Tasks

Grounded tasks (GT) have a realization and contribute to the achievement of the e-workflow goal. A GT is formally defined as follows:

$$GT(t) = <QoS, Is, Os>$$

Where *t*, *QoS*, *Is*, and *Os* are the name of the task, its QoS, a set of input parameters, and a set of output parameters, respectively. The QoS specification associated with a GT

is to be used by algorithms to synthesize the QoS of workflows based on the QoS metrics of the tasks and the Web services that compose the workflow (Cardoso, Miller *et al.* 2002).

For example, in our initial scenario, the tasks *Conference Registry, Get User Information,* and *Hotel Reservation* are grounded tasks. The GT *Conference Registry* has the following structure:

GT ("Get Conference Information") = <{time.max = 50, reliability.avg = 0.95, cost.max = 12.4, cost.max = 21.5}, {"Conference"}, {"Date", "Duration", "City"}>

Please note that the inputs and outputs in this example are associated with ontological concepts.

Service Template

When a designer needs to search for a Web service to be integrated into an e-workflow, a service template (ST) is created. A service template represents the intent of the designer to extend the functionality of an e-workflow, bringing the process closer to its ultimate goal. STs do not have a realization associated with them; they represent a structure or blueprint that the designer uses to indicate the characteristics of the Web service that is needed. A ST is specified as:

$$ST = <sn, sd, QoS, Os, Is>$$

Five fields exist: *sn*, *sd*, *QoS*, *Os*, and *Is*. The *sn* variable corresponds to the name of the Web service to be found. We will see later that the name specified does not have to syntactically match exactly with the name of the Web services to be discovered. The *sd*, *qos*, *Os,* and *Is* fields correspond to a textual description, the operational metrics, and a set of output and input parameters, respectively, of the Web service to be found.

The set of output parameters corresponds to the set of the output parameters of the tasks connected to a ST, and the set of input parameters corresponds to the set of the input parameters of the tasks the ST will be connected to. Lets us indicate the GTs to be connected to a ST with the symbol >st, and the GTs that the ST connects to with st<. Then,

$$Os = \bigcup_{gt \in st>} output(gt), Is = \bigcup_{gt \in <st} input(gt)$$

For example, our scenario contains one service template, the *Travel Reservation* template (represented by a dotted circle in Figure 2-1) that holds the following information:

ST = < "Travel_Agency", "An travel agent service that provides flight reservations based on the specification of a flight request", {cost.max=50, time.avg=5},{" Date", "Duration", "City"} ∪ {"User Name", "Address"}, {"Itinerary"}>

Service Object

The service object is a structure that holds the description of a real Web service. As stated earlier, we specified Web services semantically. A SO is formally described as follows:

$$SO = <sn, sd, QoS, Is, Os>$$

The structure is composed of five concepts: *sn*, *sd*, *QoS*, *Is*, and *Os*. The fields of a SO have the same meaning as the ones defined in a ST. This makes sense because SOs will be matched against STs.

5 Matching ST and SO

The Web service discovery and integration process is carried out by a key operation: the match function. The matching step is dedicated to finding correspondences between a service template and a service object. During the discovery phase, the match function is employed to successively match a ST against a set of SOs, which are possibly advertised in a registry (*e.g.* UDDI). The SOs are ranked based on their degree of similarity and integration with the ST. The user may then select the Web service with the highest degree of similarity and manually solve the schematic differences not already solved by the system.

We have constructed a system which implements the following idea. Given a service template and a set of service objects, the system examines the services and tries to find similarities between a ST and each SO. This is done using syntactic, operational, and semantic information as a way to increase the precision of the match. The system (1) evaluates the degree of similarity between a ST and a SO and (2) provides the means for the interoperability of services through the analysis and suggestion of connections between the SO interfaces that maximize the degree of integration with the ST.

Syntactic Similarity: The syntactic similarity of a ST and a SO is based on their *service names* and *service descriptions*. At this stage, only syntactic information is taken into account, since both fields are simply expressed using a set of words, without attaching any tag of concepts to each one.

Operational Similarity: Syntactic and semantic information allows for the selection of Web services based on their functionality, but without accounting for operational metrics. The operational similarity of a ST and a SO is calculated based on the metrics specified in their QoS model. The purpose is to determine how close two Web services are, as based on their operational capabilities.

Semantic Similarity: Purely syntactical methods that treat terms in isolation from their contexts are insufficient since they deal with syntactic but not with semantic correspondences, and since users may express the same concept in different ways (Sheth and Kashyap 1992; Lee, Kim *et al.* 1993). Therefore, we rely on semantic information to evaluate the similarity of concepts and properties that define the ST and SO interface. This evaluation will be used to calculate their degree of integration.

5.1 Syntactic Similarity Function

The syntactic similarity of a ST and a SO is calculated with the function *SynSimilarity(ST, SO)*. The similarity computation relies on the *SynNS(ST, SO)* and *SynDS(ST, SO)* functions, and the weights ω_1 and ω_2. The functions *SynNS* and *SynDS* are binary functions that compute the degree of similarity between two service names, and two service descriptions, respectively. The computation is based only on syntactical considerations, and no semantic information is taken into account at this time. Both functions return a real value between 0 and 1, indicating the degree of syntactic similarity. The weights ω_1 and ω_2 are real values between 0 and 1; they indicate the degree of confidence that the designer has in the service name and service description he supplied when constructing a ST.

$$SynSimilarty(ST, SO) = \frac{\omega_1 SynNS(ST.sn, SO.sn) + \omega_2 SynDS(ST.sd, SO.sd)}{\omega_1 + \omega_2} \in [0..1],$$

$$\text{and } \omega_1, \omega_2 \in [0..1]$$

High weight values indicate the designer's confidence in the supplied information. For example, let consider that a user is searching for a service and supplies the service name "Travel Agency" and a service description "Accepts a quote request for air travel." The user his allowed the association of a weight with the service name and with the service description. If the user is not confident about the service description given, the weight ω_2 can be set to a low value, for example 0.20. If the user is certain of the service name given, the weight ω_1 can be set to 0.8. Please note that sum of the weights does not have to add up to 1.

It is not realistic to expect that the majority of users will understand the relationship between information confidence and weighting. In view of the fact that humans often feel awkward in handling and interpreting such quantitative values (Tversky and Kahneman 1974), we have constructed a mapping table that establishes a correspondence between quantitative values and a qualitative scale (Miles and Huberman 1994). Thus, instead of explicitly specifying quantitative values, the designer can optionally select qualitative terms. An example of a mapping table (which can be customized) is expressed in Table 5-1.

Qualitative	Quantitative
Uncertain	[0.0..0.2]
Hesitant	[0.2..0.4]
Optimistic	[0.4..0.6]
Confident	[0.6..0.8]
Certain	[0.8..1.0]

Table 5-1 – Confidence Mapping Table

Several methods can be employed to match service names and descriptions. The similarity of names can be defined and measured in various ways, including equality of name, equality of canonical name representations after stemming and other preprocessing, equality of synonyms, similarity of names based on common sub-strings, pronunciation, and soundex. Service descriptions contain comments in natural language that express the intended semantics of a service. These comments can be evaluated linguistically to determine the similarity between services. The linguistic analysis can be as simple as extracting keywords from the descriptions which are used for synonym comparison, much like names, or it could be as sophisticated as using natural language-understanding technology to look for semantically equivalent expressions.

In our approach, we use "string-matching" as a way to calculate how closely service names and service descriptions resemble each other. The functions $SynNS(n_1, n_2)$ and $SynDS(d_1, d_2)$ evaluate syntactic similarity by considering the number of *q-grams* (Zamora, Pollock *et al.* 1981; Angell, Freund *et al.* 1983; Salton 1988) that their arguments have in common. To achieve a better comparison between two service descriptions we pre-process the descriptions. A common stop list is applied to remove common words with no information value such as "and" and "of" (Fox 1992); words are also reduced to their stem by removing prefixes and suffixes (Porter 1980), and duplicates are eliminated. Table 5-2 shows the results of two examples of calculating how close two Web service names are.

Service Name A	Service Name B	Result
"The Travel Agency"	"Travel Agent"	0.87
"The Travel Agency"	"An Internet Travel Agent"	0.63

Table 5-2 – Comparing Web service names

We are not so much interested in introducing a clever function for syntactic similarity, since our work focus on operational similarity, and on semantic similarity and integration, as in showing the importance of considering syntactic information during Web service discovery.

Another popular algorithm that may be considered to compare service names is the edit distance formulated by Levenshtein (1966). For the service description comparison, techniques borrowed from the information retrieval area may also be considered. For example, the frequency-inverse document frequency (Salton 1988) weighting (TF-IDF)

has been used in the LARKS system (Sycara, Lu *et al.* 1998) to match heterogeneous agents on the Internet. A very good source of information retrieval techniques can be found in Belew (2000). There is some evidence that combining different ranking methods to yield a new method can improve performance, possibly through capturing the best of the different methods (Losee 1988; Hull, Pedersen *et al.* 1996).

5.2 Operational Similarity Function

The operational similarity of a ST and a SO is calculated with the function *OpSimilarity(ST, SO)*. The binary function *OpSimilarity* computes the geometric distance of the QoS dimensions specified in the ST and the ones specified in the SO. The function returns a real value between 0 and 1, indicating the similarity of the operational metrics of its arguments. The closer to the value 1 the result is, the more similar a SO is to a ST.

$$\text{OpSimilarity}(ST, SO) = \sqrt[3]{\text{QoSdimD}(ST, SO, time) * \text{QoSdimD}(ST, SO, cost) * \text{QoSdimD}(ST, SO, reliability)}$$

The distance of two QoS dimensions is calculated using function QoSdimD(*ST, SO, dim*), where *dim* is a dimension. The function calculates the geometric distance of the distance of the individual components making up the dimension *dim* (*i.e.,* the minimum, average, and maximum value the dimension can take) of the ST and of the SO. The distance of two dimension components is called the dimension component distance (*dcd*).

$$\text{QoSdimD}(ST, SO, dim) = \sqrt[3]{\text{dcd}_{\min}(ST, SO, dim) * \text{dcd}_{\text{avg}}(ST, SO, dim) * \text{dcd}_{\max}(ST, SO, dim)}$$

Three *dcd* functions exist: $\text{dcd}_{\min}(ST, SO, dim)$, $\text{dcd}_{\text{avg}}(ST, SO, dim)$, and $\text{dcd}_{\max}(ST, SO, dim)$. The $\text{dcd}_{\min}(ST, SO, dim)$ is defined as follows:

$$\text{dcd}_{\min}(ST, SO, dim) = 1 - \frac{|\min(SO.qos(dim)) - \min(ST.qos(dim))|}{\min(ST.qos(dim))}$$

The definition of the other two functions is similar; the symbol "min" should be replaced with "avg" or "max". The functions min, avg, and max return the minimum, average, and maximum, respectively, of the QoS dimension specified in the argument.

Table 5-3 shows an example of how to compute the distance of two QoS dimensions for the time dimension. The metrics shown are from the task *Prepare Sample* from a genomics process (Cardoso, Miller *et al.* 2002). The results indicate a high similarity between the time dimension metrics of the ST and of the SO.

	Min	Avg	Max
ST	190	197	199
SO	192	196	199
dcd_x(ST, SO, *time*)	$1-\frac{\lvert 192-190\rvert}{190}$	$1-\frac{\lvert 196-197\rvert}{197}$	$1-\frac{\lvert 199-199\rvert}{199}$
QoSDimD(ST, SO, *time*)	$\sqrt[3]{\frac{188}{190}*\frac{196}{197}*1}=0.99$		

Table 5-3 – Example on how to calculate the QoS distance for the time dimension

5.3 Semantic Integration

Web service integration differs from previous work on information integration due to the number of services involved, the potential number of ontologies employed to describe service interfaces, and the polarity of input/output schema. The polarity of schema forces output schema to be connected to input schema. Furthermore, an input schema needs to have all its input parameters satisfied. This is not required for an output schema. Solutions involving a semiautomatic integration, requiring user input that defines similarities between terms or semantic interrelations (Hammer, McLeod *et al.* 1994; Kashyap and Sheth 1996; Bergamaschi, Castano *et al.* 1998) are not adequate for the Web service integration problem. It is not realistic to expect the user to provide information about potential mappings or semantic interrelations among terms for each Web service object present in a registry. We desire to develop a mechanism that automatically computes the similarity of two services, efficiently and without human intervention, and that suggests potential mappings between a ST and a SO schema which maximize their degree of integration, thus reducing structural and semantic heterogeneity.

We now present our algorithm to compute the degree of integration of a ST and a SO. This function bases its computation on the input and output parameters of both the ST and the SO.

5.3.1 Semantic Integration Function

The semantic integration function *DIntegration(ST, SO)* is a binary function that returns the degree of integration between its operators. The operands are a service template (ST) and a service object (SO), and the result is a real value between 0 and 1.

$$DIntegration(\text{ST, SO}) \in [0..1]$$

The underlying goal of the function is to establish a mapping between the output of the ST (*ST.O*) and the input of the SO (*SO.I*) and a mapping between the output of the SO (*SO.O*) and the input of the ST (*ST.I*) that maximize the degree of integration.

Depending on the data present in a service template, four distinct cases can occur when comparing input and output parameters. The definition of the function *DIntegration* captures these four cases.

$$DIntegration(ST, SO) = \begin{cases} \dfrac{\dfrac{\Pi(ST.Os, SO.Is)}{|SO.Is|} + \dfrac{\Pi(SO.Os, ST.Is)}{|ST.Is|}}{2}, & ST.Os \neq \varnothing, ST.Is \neq \varnothing \\ \Pi(ST.Os, SO.Is)/|SO.Is|, & ST.Os \neq \varnothing, ST.Is = \varnothing \\ \Pi(SO.Os, ST.Is)/|ST.Is|, & ST.Os = \varnothing, ST.Is \neq \varnothing \\ 0, & ST.Os = \varnothing, ST.Is = \varnothing \end{cases}$$

The simplest case occurs when a ST does not specify any inputs or outputs. In this case, the integration degree is evaluated to 0. If a ST only specifies a set of outputs and no inputs, then the function Π(*Os, Is*) is employed to compute the semantic mapping between the outputs *Os* of the ST and the inputs *Is* of the SO. The result of applying the function Π is normalized with respect to the number of inputs being mapped. The rationality of this normalization is that when matching *n* outputs of a task *a* against *m* inputs of a task *b*, we are interested in satisfying all the input requirements of task *b*. A task or Web service always needs to have its mandatory inputs satisfied with data in order to correctly carry out its intended function. Optional inputs are not taken into account. Nevertheless, a designer may explicitly mark an optional input as mandatory if he wishes optional inputs to be considered during the integration evaluation process. The same concept is applied if the ST includes inputs but no outputs.

Finally, if a ST includes both a set of outputs and a set of inputs the mapping function Π is applied to both sets. In this case, we compute the arithmetic mean of the normalized results from the evaluation of function Π. We use the arithmetic mean because we give the same importance to the normalized semantic mapping of the ST outputs with the SO inputs and the normalized semantic mapping between SO outputs with ST inputs.

5.3.2 Mapping Inputs and Outputs

The function Π(*Os, Is*), where *Os* is a set of output parameters and *Is* a set of input parameters, computes the best mapping that can be obtained from connecting the outputs of the set *Os* to the inputs of set *Is*.

$$\Pi(Os, Is) = \begin{cases} Max(\Pi(Os - O, Is - I) + \pi(O, I)), & Os \neq \varnothing, Is \neq \varnothing, O \in Os, I \in Is \\ 0, & Os = \varnothing \vee Is = \varnothing \end{cases}$$

Please note that the number of mappings established is *Min(|Os|, |Is|)*. Each output *O* of *Os* is matched against each input *I* of *Is*. Their semantic similarity degree is evaluated with function $\pi(O, I)$. Since input/output parameters are associated with ontological concepts (see section 3.2), the function $\pi(O, I)$ compares two concept classes represented by *O* and *I*.

$$\pi(O, I) = \begin{cases} SemS'(O, I), & \Omega(O) = \Omega(I) \\ SemS''(O, I) / \mid p(I) \mid, & \Omega(O) \neq \Omega(I) \end{cases}$$

The function $\pi(O, I)$ takes into consideration the ontology(ies) associated with the concepts being compared. If the concepts are from the same ontology, *i.e.* $\Omega(O) = \Omega(I)$, the function *SemS'*(*O, I*) is employed to evaluate their similarity; otherwise, if they are from distinct ontologies, *i.e.* $\Omega(O) \neq \Omega(I)$, the function *SemS''*(*O, I*) is used. We make this distinction since the information available when comparing concept classes from the same ontology has a different nature and structure which is not present when comparing concepts from distinct ontologies. The result of function *SemS''* is normalized with respect to the number of properties of the input concept *I*. As we will see, the evaluation of the similarity of two concepts is based on their composing properties. Once again, the reason for this normalization is to obtain a measure that reflects the fact that all the mandatory input properties need to have an output property associated with them in order for a task or Web service to work properly.

5.3.3 Comparing Outputs and Inputs from the same Ontology

The function *SemS'(O, I)* evaluates the similarity of two concept classes associated with an output (*O*) and an input (*I*), conceptualized within the same ontology. Four distinct scenarios can occur: a) the concepts are the same (*O*=*I*), b) the concept *I* subsumes concept *O* (*O*>*I*), c) the concept *O* subsumes concept *I* (*O*<*I*), or d) concept *O* is not directly related to concept *I* (*O*≠*I*). In the latter case, the concept *O* does not have a parent/child relationship with concept *I*, but both concepts have a parent concept in common.

$$SemS'(O,I) = \begin{cases} 1, & O = I \\ 1, & O > I \\ \dfrac{|\,p(O)\,|}{|\,p(I)\,|}, & O < I \\ Similarity'(O,I), & O \neq I \end{cases}$$

In the first case, which is the simplest, if the two concepts are equal then intuitively their similarity is maximal; therefore, it is evaluated to one. In the second case, if the concept *I* subsumes the concept *O*, their similarity is also evaluated to 1. The similarity is maximal since if an output represented with a concept *O* is a subclass of an input represented with a concept *I* it has at least the same set of properties as *I*. Thus, all input properties have a corresponding output property associated with them. In the third case, the concept *O* subsumes the concept *I* (*O*<*I*). As a result, some properties of the concept *I* may not have an output property associated with them. The similarity is set to the ratio of the number of properties of concept *O* (represented with |*p*(*O*)|) and the number of properties of concept *I* (|*p*(*I*)|). This ratio indicates the percentage of input properties of the SO that are satisfied by output properties of the ST.

In the last case, the concepts *O* and *I* are not equal and do not subsume each other in any way. In this case, assessing similarity is a judgment process that requires two "things" to be decomposed into elements in which they are the same and into elements in which they are different (Tversky 1977). Assessing the similarity of concepts is an important process for systems such as information retrieval and information integration. A number of approaches to measuring conceptual similarity between words have been taken in the past. Tversky's feature-based similarity model (Tversky 1977) has been considered as the most powerful similarity model to date (Richardson and Smeaton 1995).

Tversky introduced a general feature-counting metric for similarity called the feature-contrast model. This model is based on the idea that common features tend to increase the perceived similarity of two concepts, while feature differences tend to diminish perceived similarity. Tversky's model claims that feature commonalities tend to increase perceived similarity more than feature differences can diminish it. That is, when assessing similarity we give more credence to those features that concepts have in common than to those that distinguish them. For instance, a SUV (Sport Utility Vehicle) and a sedan are similar by virtue of their common features, such as wheels, engine, steering wheel, and gears, and are dissimilar by virtue of their differences, namely height and the size of the tires. Based on Tversky's model, we introduce a similarity function based on the number of properties shared among two concepts c_1 and c_2. Our similarity function is defined as followed, where the function *p(x)* retrieves all the properties associated with a concept *a* and function |*s*| corresponds to the number of elements in the set *s*.

$$similarity'(O,I) = \sqrt{\frac{|\,p(O) \cap p(I)\,|}{|\,p(O) \cup p(I)\,|} * \frac{|\,p(O) \cap p(I)\,|}{|\,p(I)\,|}}$$

The *similarity'*(*O*,*I*) function computes the geometric distance between the similarity of the domains of concept *O* and concept *I* and the ratio of matched input properties from the concept *I*.

As an example, let us illustrate the use of function *SemS'(O, I)* for the four cases – a), b), c) and d) – that can occur when connecting an output *O* to an input *I* (see Figure 5-1). In our example, both input and output are conceptualized with concepts from the same ontology, *i.e.* $\Omega(O) = \Omega(I)$ = Time ontology (an example using difference ontologies is given in the next section). The time ontology is not fully represented in Figure 5-1; only the concepts that are employed in our example are shown.

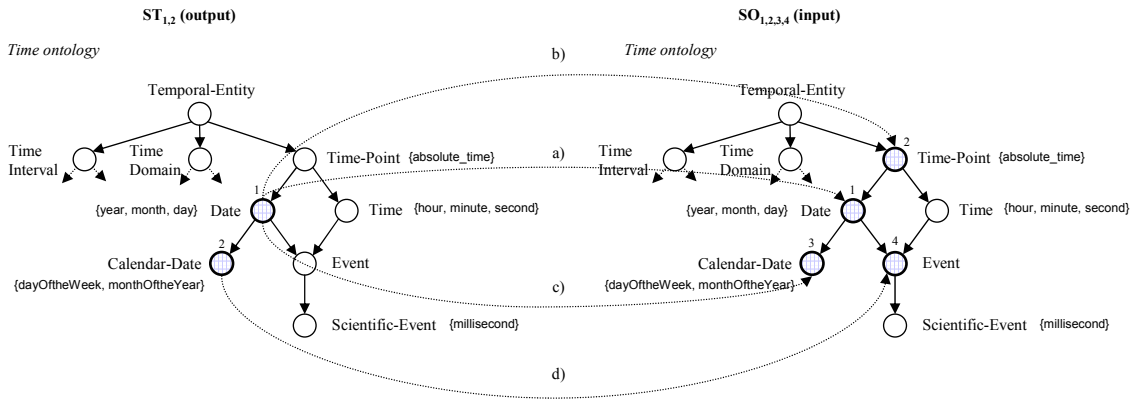

Figure 5-1 – Comparing concepts from the same ontology

	Service Template	Output		Service Object	Input
a)	ST_1	Date (1)	→	SO_1	Date (1)
b)	ST_1	Date (1)	→	SO_2	Time-Point (2)
c)	ST_1	Date (1)	→	SO_3	Calendar-Date (3)
d)	ST_2	Calendar-Date (2)	→	SO_4	Event (4)

Table 5-4 – The four examples illustrated in Figure 5-1.

The four cases that may occur are listed in Table 5-4 and are evaluated as follows:

- In case a), both *O* and *I* are associated with the same concept (*Date*). This is the simplest case. Since the output of the ST_1 matches perfectly the input of the SO_1 the similarity is evaluated to 1.

- In case b), the output *O* is associated with the concept *Date*, and the input *I* is associated with the concept *Time-Point*. Since the concept *Time-Point* subsumes the concept *Date*, the properties of the concept *Date* (the set {absolute_time, year, month, day}) is a superset of the properties of the concept *Time-Point* (the set {absolute_time}). Therefore, the output *O* of the ST_1 can be connected to the input *I* of the SO_2 without any property of *I* being left unfulfilled; there is a direct semantic correspondence and value mapping. All the properties of *I* exist in *O*. As a result, the similarity is evaluated to 1.
- In case c), the output *O* is associated with the concept *Date* and the input *I* is associated with the concept *Calendar-Date*. Since the concept *Date* subsumes concept *Calendar-Date*, the properties of the concept *Date* (the set {absolute_time, year, month, day}) is a subset of the properties of the concept *Calendar-Date* (the set {dayOftheWeek, monthOftheYear, absolute_time, year, month, day}). In this case, when the output *O* is connected to the input *I* some properties of *I* are left unfulfilled (the properties dayOftheWeek and monthOftheYear). To indicate this mismatch the similarity is set to the ratio of the number of properties of *O* and the number of properties of *I*, which in this case is $|p(O)|/|p(I)| = 4/6 \approx 0.67$.
- In the last case (d), the output *O* of the ST_2 is associated with the concept *Calendar-Date* and the input *I* of the SO_4 is associated with the concept *Event*. The concept *Event* has the set of properties {absolute_time, year, month, day, hour, minute, second} and the concept *Calendar-Date* has the set of properties {dayOftheWeek, monthOftheYear, absolute_time, year, month, day}. Since the concepts do not have a parent/children relationship, the function *similarity'*(*O*,*I*) is used to compute the geometric distance between the similarity of the domains of concept *Calendar-Date* and concept *Event* and the percentage of input properties that are fulfilled with an output property from *O*. The similarity is evaluated as follows:

$$s_1 = p(CalendarDate) = \{\text{dayOftheWeek, monthOftheYear, absolute\_time, year, month, day}\}$$

$$s_2 = p(Event) = \{\text{absolute\_time, year, month, day, hour, minute, second}\}$$

$$s_3 = p(CalendarDate) \cap p(Event) = \{\text{absolute\_time, year, month, day}\}$$

$$s_4 = p(CalendarDate) \cup p(Event) = \{\text{dayOftheWeek, monthOftheYear, absolute\_time, year, month, day, hour, minute, second}\}$$

$$similarity'(CalendarDate, Event) = \sqrt{\frac{|s_3|}{|s_4|} * \frac{|s_3|}{|s_2|}} = \sqrt{\frac{4}{9} * \frac{4}{7}} \approx 0.504$$

The result of evaluating the function *similarity'*(*Calendar-Date*, *Event*) indicates a low degree of integration between the concepts *Calendar-Date* and *Event*. On one hand,

the concepts show a low similarity according to the feature-contrast model (4/9). On the other hand, only four out of the seven input properties are connected to output properties.

5.3.4 Comparing Outputs and Inputs from Distinct Ontologies

The problem of determining the similarity of concepts defined in different ontologies is related to the work on multi-ontology information system integration. When the input and output concepts to compare are from distinct ontologies, the evaluation of their similarity is more complex. Our approach for this problem uses the same rationale that we have exploited earlier to compare input and output concepts from the same ontology without any parent/child relationship. Additionally, we also take into account syntactic similarities among concepts.

Since we compare input and output concept classes based on their properties, the first step is to find the best mapping between output and input concept properties. This objective is achieved using the function *S*emS''(*O, I*), which is very similar to function *ΠΠ(Os, Is)* previously defined as being alble to find the best mapping between a set of outputs and a set of inputs.

$$SemS''(O,I) = \begin{cases} Max(SemS''(O-o, I-i) + S(o,i)), & O \neq \varnothing, I \neq \varnothing, o \in O, i \in I \\ 0, & O = \varnothing \vee I = \varnothing \end{cases}$$

Each property *o* of output *O* is mapped with a property *i* of input *I*. A property *o* is associated with a property *i* that maximizes the semantic similarity computed, using the function *S(o, i)*.

The function *S*(*o, i*) calculates the similarity between a property *o* and a property *i*. Three distinct cases are considered: (1) the ontological properties involved are associated with a primitive data type (see section 3.2), (2) the properties are associated with concept classes, and (3) one property is associated with a primitive data type, while the other is associated with a concept class. The function *S*(*o, i*) is shown below.

$$S(o,i) = \begin{cases} \sqrt[3]{SemDS(d(o),d(i)) * SynS(n(o),n(i)) * SemRS(r(o),r(i))}, & o \text{ and } i \text{ are primitive types} \\ SemDS(o,i), & o \text{ and } i \text{ are concept classes} \\ f(o,i), & \text{otherwise} \end{cases}$$

In the first case, the similarity of the properties is computed based on the geometric distance of (a) the semantic similarity of their domains (*i.e.*, concept classes), (b) the syntactic similarity of their names, and (c) the semantic similarity of their ranges.

a). The semantic similarity of the domains of two properties, *d*(*o*) and *d*(*i*), is evaluated using function *SemDS(od, id)*, which is based on Tversky's model.

$$SemDS(od, id) = \frac{|\, p(od) \cap p(id) \,|}{|\, p(od) \cup p(id) \,|}$$

When calculating the intersection of sets $p(od)$ and $p(id)$, two elements intersect if their syntactic similarity, using the *q-grams* methodology (see section 5.1), is greater than a constant c (we are currently using $c = 0.75$).

b). The syntactic similarity of property names is calculated using the function $SynS(n_1, n_2)$. This function uses *q-grams* to determine the similarity of two property names.

c). The semantic similarity of the ranges of two properties, $r(o)$ and $r(i)$, is evaluated using the function $SemRS(r(o), r(i))$ defined below.

$$SemRS(or, ir) = \begin{cases} 1, & or = ir \\ 1, & or = \text{integer}, ir = \text{string} \\ 2/3, & or = \text{long}, ir = \text{integer} \\ 1/3, & or = \text{double}, ir = \text{integer} \\ 1, & or = \text{integer}, ir = \text{long} \\ 0, & \text{otherwise} \end{cases}$$

The function *SemRS*(*or, ir*) indicates the validity and the integration degree that is obtained when an output with a primitive data type dt_a is connected to a particular input with a primitive data type dt_b. This function is automatically created based on the capabilities of the WfMS where the e-workflow being constructed will be enacted. A workflow system that has the competence of making data type conversions (*i.e.*, converting one data type into another) on the data exchanged among tasks can formally define and describe this ability with the customization of function *SemRS*.

For example, if a WfMS can map an output property of task *a*, with range *integer*, to an input property of task *b*, of range *long*, this can be indicated by adding the following entry to function *SemRS*:

1, *or=integer* and *ir=long*

The similarity is maximal, and it is set to 1, since the WfMS can map an *integer* data type to a *long*. When an association between two data types is not valid, the function *SemRS* returns 0. In other situations, it is possible to specify a fuzzy degree of integration by setting the similarity to a value greater than zero and less than one. For example, let us consider the following entry:

1/3, *or=double* and *ir=integer*

In this case, the WfMS is able to perform a specific data type conversion (*double* to *integer*), but the conversion is not preferred or recommended since a loss of information may occur.

In the second case (2) of function *S(o, i)*, since *o* and *i* are concept classes, we use the function *SemDS*(*o*, *i*) to compute their similarity. The function *SemDS* evaluates the similarity of two concept classes only in a shallow fashion. An alternative is to use a deep-based similarity function (*i.e.*, recursively compare subclasses). This can be achieved by substituting the function *SemDS(o, i)* present in function *S(o, i)* with the function *SemS''*(*od, id*)/|*p*(*id*)|.

In the third case (3), function *f*(*o, i*) is used to calculate the similarity among a property associated with a basic data type and a property associated with a data class. For the definition of this function we rely on the concept of dynamic attributes that has been proposed in (Litwin and Abdellatif 1986) to specify the mappings between different attributes. The idea is to define a function or a set of functions that indiate the possible mappings between a property and a concept class. Examples of such mappings can be found in (Kashyap and Sheth 1993).

Let us illustrate the use of functions *SemS''*(*O, I*) and *S*(*o, i*) with the example shown in Figure 5-2.

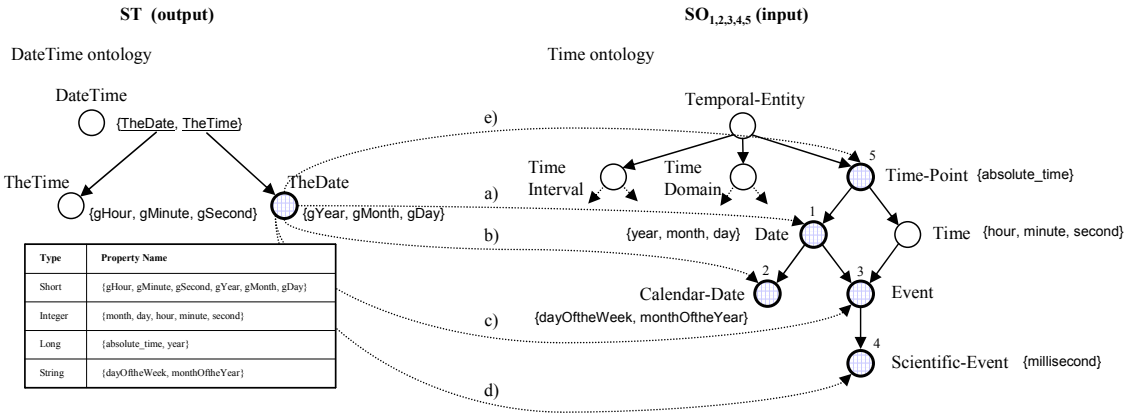

Figure 5-2 – Comparing proprieties referencing primitive data types.

To makes the example easier to understand, the ST employed to find a SO only specifies a set of outputs, with no inputs. Furthermore, we carry out the computation of function *SemS''*(*O, I*) for only one of the outputs of the ST (the *TheDate* parameter) and for only one of the SO inputs (the inputs are represented with the indexes 1 through 5 in Figure 5-2). We consider that five SOs ($SO_{1, 2, 3, 4, \text{and } 5}$) are present in the registry during the discovery procedure. The five cases are shown in Table 5-5.

	Service Template	Output		Object Template	Input
a)	ST	TheDate	→	SO_1	Date
b)	ST	TheDate	→	SO_2	Calendar-Date
c)	ST	TheDate	→	SO_3	Event
d)	ST	TheDate	→	SO_4	Scientific-Event
e)	ST	TheDate	→	SO_5	Time-Point

Table 5-5 – The five examples illustrated in Figure 5-2.

The SO_1 input is associated with the class concept *Date*. The SO_2 input is associated with the class concept *Calendar-Date*. The SO_3 input is associated with the class concept *Event*. Finally, the SO_4 and SO_5 inputs are associated with the concept class *Scientific-Event* and *Time-Point*, respectively.

During the discovery process, the ST is compared with each SO individually. Therefore, the function *SemS''*(*O, I*) is applied five times. In Figure 5-2, the computation of the function between the output of a ST and the input of a $SO_{1..5}$ is represented with a letter (a, b, c, d, or e).

Let us start with the computation of function *SemS''*(*O, I*) to evaluate the degree of integration of the concept class *TheDate* (from the *DateTime* ontology, *i.e.*, the concept Ω(DateTime).TheDate) and the concept class *Calendar-Date* (from the *Time* ontology, *i.e.*, the concept Ω(Time).Calendar-Date). Figure 5-3 shows the mappings carried out by function *SemS''*(*TheDate, Calendar-Date*).

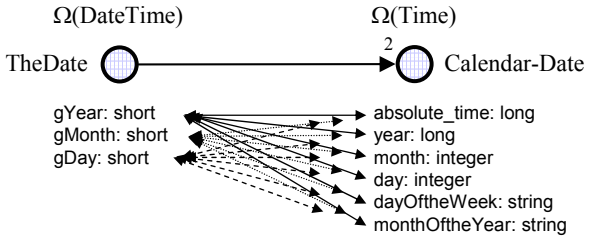

Figure 5-3 – Evaluating the degree of integration

For each connection shown in Figure 5-3, function *S*(*o, i*) is called on to evaluate the degree of integration among two properties. Since in our example the output and input properties of the concept classes *O* and *I* reference primitive data types, function *S* will uniquely use the case (1) described previously. This corresponds to the use of the following function:

$$\sqrt[3]{SemDS(d(o),d(i)) * SynS(n(o),n(i)) * SemRS(r(o),r(i))}$$

Let us trace the computation of *S*(*o, i*) with *o* = ”*gDay*” and *i* = ”*day*”. The function *SemDS* evaluates the similarity of the domains (concept classes) of properties *o* and *i*. The properties ”*gDay*” and ”*day*” have the domain concepts *TheDate* and *Calendar-Date*,

respectively, *i.e., d*(*gDay*) = *TheDate* and *d*(*day*) = *Calendar-Date*. Therefore, *SemDS*(*TheDate, Calendar-Date*) is evaluated the following way:

$$p(\textit{TheDate}) = \{\text{gMonth}, \text{gYear}, \text{gDay}\}$$

$$p(\textit{Calendar-Date}) = \{\text{absolute\_time}, \text{year}, \text{month}, \text{day}, \text{dayOfTheWeek}, \text{monthOfTheYear}\}$$

$$\textit{SemDS(TheDate, Calendar-Date)} = \frac{|\ p(\textit{TheDate})\ \cap p(\textit{Calendar} - \textit{Date})\ |}{|\ p(\textit{TheDate})\ \cup p(\textit{Calendar} - \textit{Date})\ |} = 0.5$$

This result, 0.5, indicates that the domains of properties *o* and *i* are somewhat similar, which follows our perception that the concepts *TheDate* and *Calender-Date* are similar.

The second function to be evaluated is *SynS*(*no, ni*). This function computes the syntactic similarity of the property names *no* and *ni*. In our example, the similarity of properties *gDay* and *day* is evaluated to 0.8. This result indicates a close syntactic similarity between the two property names. Other examples of the application of the function *SynS* for the properties involved in our example are:

$$\textit{SynS}(\textit{gDay, dayOfTheWeek}) = 0.29$$

$$\textit{SynS}(\textit{gMonth, monthOfTheYear}) = 0.44$$

The last function to be evaluated is function *SemRS*(*r*(*o*), *r*(*i*)), which calculates the similarity of the ranges of properties *o* and *i*. For the properties *gDay* and *day*, the following metric is obtained:

$$\textit{SemRS}(r(\textit{gDay}), r(\textit{day})) = \textit{SemRS}(\textit{short, integer}) = 1.0$$

This result indicates that the workflow system that will be enacting the process being constructed supports the connection of parameters of type *short* to parameters of type *integer*. An example of a connection among properties not supported or desired is the following one:

$$\textit{SemRS}(r(\textit{gDay}), r(\textit{dayOfTheWeek})) = \textit{SemRS}(\textit{short, string}) = 0.0$$

Having calculated the functions *SemDS*, *SynS*, and *SemRS*, we can now compute function *S*. The result of evaluating *S*(*gDay, day*) is,

$$\sqrt[3]{0.5 * 0.8 * 1.0} = 0.74$$

Table 5-6 shows the results of applying function $S(o, i)$ to various properties of the concept classes *TheDate* and *Calendar-Date*.

o	*i*	*SemDS*	*SynS*	*SemRS*	*S*
gMonth	dayOfTheWeek	0.5	0.12	0.0	0.0
gYear	monthOfTheYear	0.5	0.35	0.0	0.0
gDay	month	0.5	0.0	1.0	0.0
gDay	year	0.5	0.0	1.0	0.0
gDay	day	0.5	0.8	1.0	0.74
gDay	time	0.5	0.0	1.0	0.0
gDay	monthOfTheYear	0.5	0.0	0.0	0.0
gYear	year	0.5	0.86	1.0	0.75
gMonth	monthOfTheYear	0.5	0.44	0.0	0.0
gMonth	month	0.5	0.89	1.0	0.76

Table 5-6 – Examples of the evaluation of function *S*(*o*, *i*).

Once all the possible mappings between the properties of the output concept class TheDate and the input concept class Calendar-Date are evaluated, the function SemS''(TheDate, Calendar-Date) returns the result shown in Table 5-7 line b). The table also shows the results for all the five cases initially considered in Figure 5-2.

	ST	*O*	SO	*I*	*SemS''*(*O, I*)
(a)	ST	TheDate	SO_1	Date	2.58
(b)	ST	TheDate	SO_2	Calendar-Date	2.25
(c)	ST	TheDate	SO_3	Event	2.14
(d)	ST	TheDate	SO_4	Scientific-Event	2.05
(e)	ST	TheDate	SO_5	Time-Point	0.00

Table 5-7 – Example of computing function *SemS''*(*O,I*).

The function *SemS''*(*O, I*) returns the cumulative degree of similarity of the mappings between two concept classes, *i.e.* it corresponds to the sum of the weights associated with a bipartite graph constructed with the properties of the output concept and the properties of the input concept. While the function *SemS''*(*O, I*) gives some information relatively to the possible mapping of two input and output parameters, the function does not take into account the number of mandatory inputs that need to be satisfied. Thus, a more significant and useful piece of information is the result obtained from applying function $\pi(O, I)$, which normalizes function *SemS''*(*O, I*) with respect to the number of input properties of the concept class *I*. The results of applying function $\pi(O, I)$ to our example is shown in Table 5-8.

	ST	*O*	SO	*I*	π(*O, I*)
(a)	ST	TheDate	SO_1	Date	0.65
(b)	ST	TheDate	SO_2	Calendar-Date	0.38
(c)	ST	TheDate	SO_3	Event	0.31
(d)	ST	TheDate	SO_4	Scientific-Event	0.26
(e)	ST	TheDate	SO_5	Time-Point	0.00

Table 5-8 – Example of computing function π(*O, I*).

It can be seen that function π(*O, I*) returns values closer to 1, when the concept classes being compared exhibit a higher degree of similarity. This is the case for the concepts Ω(DateTime).TheDate and Ω(Time).Calendar-Date. When two concepts are not similar the function returns 0, which is the case for the concepts Ω(DateTime).TheDate and Ω(Time).Time-Point.

5.3.5 Mapping Outputs with Inputs

While the algorithm presented does not explicitly show how the mapping between the outputs and inputs of two services which maximize the degree of integration is constructed, this is achieved by keeping track of the best mapping obtained when computing function *ΠI(Os, Is)* and function *SemS''*(*O, I*).

6 System Architecture

The core of our work has already been presented in the previous section, with the description of the algorithm to match a ST against a set of SOs. Therefore, in this section we will only briefly describe the architecture of our prototype. Our system is composed of two main services: registry service and discovery service, as illustrated in Figure 6-1.

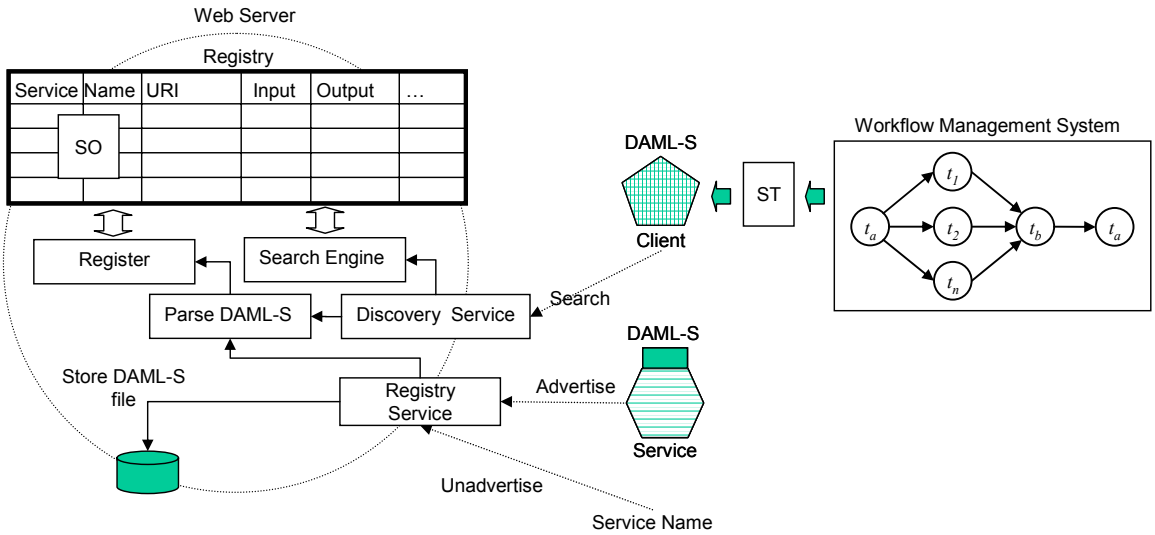

Figure 6-1– System Architecture

The services available (registry and discovery services) to users and to the WfMS are both implemented using *servlets* and are accessible through HTTP. We are considering extending the access to allow RMI calls.

Suppliers access the registry service to advertise and unadvertise their Web services. To make an advertisement, a supplier registers a DAML-S service object (SO) with the system. To unadvertise a service, the only information necessary is the name of the service.

Clients and customers typically access the system to find Web services previously registered (Figure 6-2). This is achieved by sending a service template (ST) to the system. The service template specifies the requirements about the service to discover. Service templates are described using DAML-S, more precisely by using the *profile.daml* ontology (see section 3.3).

Figure 6-2 – The Web Service Discovery page.

Once the system receives an advertisement or a discovery message, the SO or the ST received are parsed, using the Jena toolkit (Jena 2002). The Jena DAML API provides support for loading DAML ontologies onto Jena RDF models. Since DAML-S parsers are not yet available, we have built a very simple DAML-S parser on top of the Jena Toolkit to extract specific information from service objects – such as service name, service description, QoS model, inputs, and outputs. The syntactic, operational, and semantic similarity functions will make use of this information.

The information retrieved from parsing a service advertisement is stored in a registry (

Figure 6-1). The registry is a service capability table, where service descriptions are added or removed in response to advertised and unadvertised messages. The registry table and its contents are stored in physical memory for fast access.

When the system receives a discovery message (*i.e.* a ST) from a workflow system for example, it is parsed and matched against the set of SOs registered. The results are ranked according to the criteria specified when the ST was sent to the system (Figure 6-2). Three ranking methods are available, based on: syntactic, semantic, and operational metrics. Better matches are characterized by a score closer to 1. Finally, the ranked candidates are returned to the entity that issued the query. Figure 6-3 shows the results of a query. For each SO present in the registry, a detailed information sheet comparing it against the ST is constructed. It includes the results of evaluating the SO against the ST: syntactically, based on operations, and semantically. Finally, it also includes the suggested data mappings between the ST and the SO (which outputs should be connected to which inputs). This is illustrated in Figure 6-3.

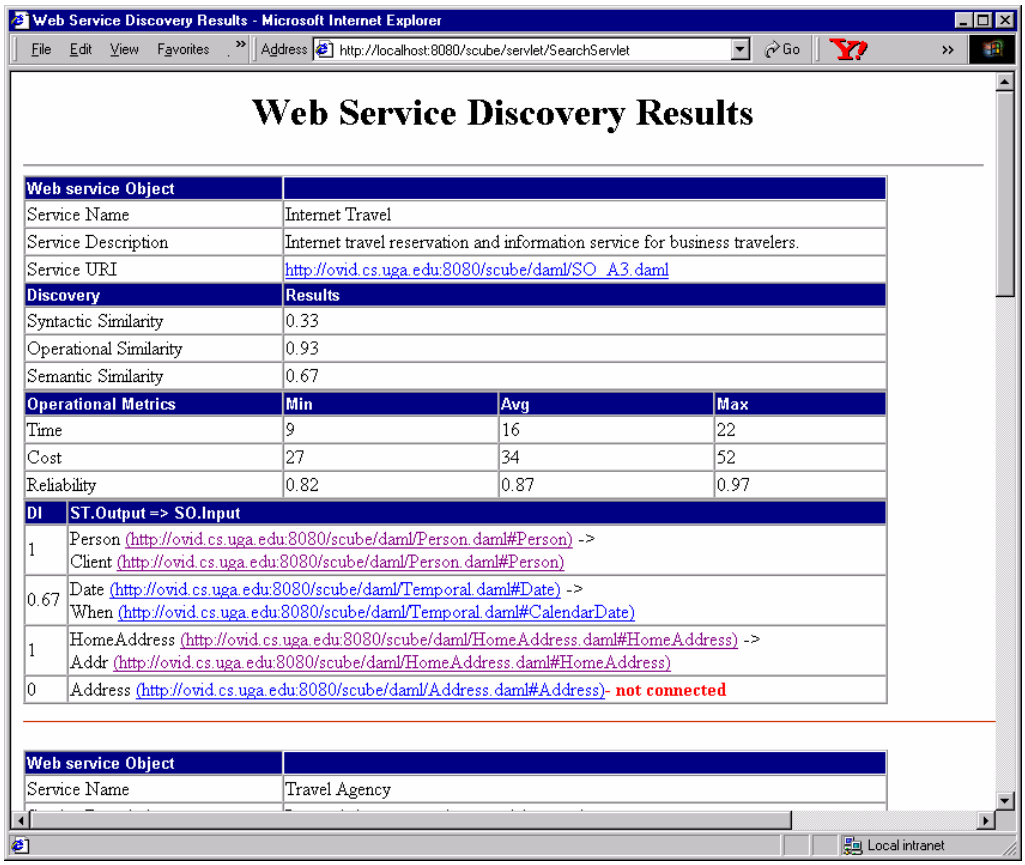

Figure 6-3 – Web Service Discovery Results page

7 Related work

Our work is directly related to ontology-based Web service discovery, search, match, and integration, and indirectly related to information retrieval systems and information integration systems.

The work that most closely relates to ours is described in Paolucci, Kawamura *et al.* (2002). They present an algorithm that deals with the localization of Web services, but

they do not address the interoperability problem. Their system also uses the service profile ontology from the DAML-S specification language. Their work considers only the matching of input/output concepts defined by the same ontology. When input/output concepts from different ontologies are matched, the algorithm simply evaluates the match to "fail". This is a limitation. Web services are heterogeneous and autonomous by nature; therefore it is advantageous to compare outputs and inputs that subscribe to different ontologies. The similarity function described is based on the taxonomy of the ontology, accounting for the parent/child relationship between concepts. If the concepts associated with an output and with an input do not have a parent/child relationship the algorithm evaluates the match to a "fail" and does not try to assess a degree of similarity. The algorithm uses the minimal distance between concepts in the taxonomy tree. We believe that a feature-based approach rather than one employing the taxonomy of the ontology achieves better precision in the discovery process. What makes two concepts distinct is not always their distance in a taxonomy tree, but the number of properties in which they are the same and in which they are different. As a last difference, operational metrics of Web services are not taken into account when discovering services.

González-Castillo, Trastour *et al.* (2001) also use DAML+OIL to semantically describe Web services; their approach to the matchmaking of services is based on a subsumption tree. Their algorithm follows a very similar approach to the one taken by Paolucci, Kawamura *et al.* (2002), in the sense that it only accounts for relationships among concepts defined within the same ontology. Their system does not use DAML-S for the description of Web services (the system was developed before its existence). Instead, they have developed their own specification for Web service description, but no notion of inputs and outputs was defined. As a result, the matching of Web services is carried out based on service description, not accounting for inputs and outputs. Their approach does not target the discovery of Web services based on operational metrics, nor does it deal with the Web service integration problem.

Another approach that also uses a specific language to describe service advertisements and requests is the LARKS (Language for Advertisement and Request for Knowledge Sharing) system (Sycara, Klusch *et al.* 1999). The LARKS language can be seen as a precursor of the DAML-S specification. The system uses ontologies defined by a concept language (ITL). Their approach does not provide an automatic solution for the computation of the similarity of concepts defined in distinct ontologies. Furthermore, the technique used to calculate the similarity of ontological concepts involves the construction of a weighted associative network, where the weights indicate the belief in relationships. While they argue that the weights can be set automatically by default, it is clear that the construction of realistically weighted relationships requires human involvement, which becomes an hard task when thousands of agents are available. Their work does not consider the matchmaking of agent-based operational metrics. While the output and input parameters of agents are compared using syntactic and semantic matching methods, the algorithm presented does not supply a mapping of potential connections between the outputs and inputs of two agents that yields a maximum degree of integration.

In the information retrieval area, Bejamins and Fensel (1998) present the (KA)2 system, an ontology-based information retrieval system for the World-Wide Web. The system allows a community to build a knowledge base collectively, based on consensual

knowledge, by populating a shared ontology. Using the shared ontology, a web-crawler accesses the web pages and uses the ontology to infer answers. The FindUr project (McGuinness 1998) is another initiative in ontology-based information retrieval on the Web. Their work focuses on query expansion and online searching. The use of ontologies has been shown to improve the search from the perspectives of recall and precision, as well as ease of query formation. The OntoSeek (Guarino, Masolo *et al.* 1999) project has also shown that ontologies improve content-based searches. Their work focuses on specific classes of information repositories: yellow pages and product catalogues. Richardson and Smeaton (1995) introduce an approach to information retrieval based on computing a semantic distance measurement between concepts or words and using this word distance to compute the similarity between a query and a document.

Ontologies have been employed as a common basis for information integration. Ontologies allow for the modeling of the semantic structure of individual information sources, as well describing models of a domain that are independent of any particular information source. Several systems have been developed using this solution. Projects include Carnot (Woelk, Cannata *et al.* 1993), InfoSleuth (Bayardo, Bohrer *et al.* 1997), SIMS (Arens, Hsu *et al.* 1996), OBSERVER (Mena, Kashyap *et al.* 1996; Kashyap and Sheth 1998), and COIN (Bressan, Fynn *et al.* 1997). These projects differ from our work in their reduced number of ontologies involved in the integration process and due to their approaches, which require user involvement to manually resolve differences among ontologies. Additionally, a vast amount of the work done is directed to solve schematic differences in multidatabase systems; this does not face the schema polarity problem.

8 Conclusions

In this paper we have presented a set of challenges that the emergence of Web services and e-services has brought to organizations. While in some cases Web services may be utilized in an isolated form, it is normal to expect Web services to be integrated as part of workflows processes. This entails research in two areas. Mechanisms to efficiently discover Web services during an e-workflow (*i.e.*, a workflow managing traditional tasks and Web services) composition process and to facilitate their subsequent integration with the e-workflow host.

We present a methodology and a set of algorithms for Web service discovery based on three dimensions: syntax, operational metrics, and semantics. This approach allows for Web service discovery not only based on functional requirements, but also on operational metrics.

The need to discover workflow components based on operational metrics has a greater importance when Web services are involved, as compared to workflow tasks. The autonomy of Web services does not allow for users to identify their operational metrics at design time, or prior to their actual execution. The development of mechanisms for the discovery of Web services based on operational metrics allows organizations to translate their vision into their business processes more efficiently, since e-workflows can be designed according to QoS requirements, goals, and objectives.

To facilitate the discovery and posteriori integration of Web service into workflows we propose an approach based on the use of ontologies to describe workflow tasks and Web service interfaces. Ontology-based approaches have already proved to be an

important solution to information integration in order to achieve interoperability. During an e-workflow composition, there is a loss of information associated with Web service task interfaces because a large part of the domain knowledge a developer employs when deploying a Web service is not present at composition time.

In our work we have devised an algorithm and implemented a prototype to discover and facilitate the resolution of structural and semantic differences during the integration process with an e-workflow. The algorithm uses a feature-based model to find similarities across workflow tasks and Web service interfaces. The system determines and evaluates the best mapping between the outputs and inputs of a SO and the workflow host that yields the highest degree of integration.

Acknowledgements: We would like to acknowledge Abhijit Patil, Ruoyan Zhang, and Swapna Oundhakar for developing an earlier version of the prototype presented in this work.

9 References

Angell, R. C., G. E. Freund and P. Willett (1983). "Automatic spelling correction using a trigram similarity measure." Information Processing and Management **19**(4): 255-161.

Ankolekar, A., M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara and H. Zeng (2001). DAML-S: Semantic Markup for Web Services. Proceedings of the International Semantic Web Working Symposium (SWWS). pp. 39-54.

Arens, Y., C. Hsu and C. A. (1996). Knoblock Query Processing in the SIMS Information Mediator. Menlo Park, CA, AAAI Press.

Arpinar, I. B., J. A. Miller and A. P. Sheth (2001). An Efficient Data Extraction and Storage Utility for XML Documents. Proceedings of 39th Annual ACM Southeast Conference, Athens, GA. pp. 293-295.

Barbar, D., S. Mehrothra and M. Rusinkiewicz (1996). "INCAs: Managing Dynamic Workflows in Distributed Environments." Journal of Database Management **7**(1): 5-15.

Bayardo, R. J., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh and D. Woelk (1997). InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM Press, New York. pp. 195-206.

Belew, R. K. (2000). Finding Out About : A Cognitive Perspective on Search Engine Technology and the WWW. Cambridge, U.K, Cambridge University Press.

Benjamins, V., D. Fensel and A. Prez (1998). Knowledge Management through Ontologies. Proceedings of the Second International Conference on Practical Aspects Knowledge Management. pp. 5.1-5.12.

Bergamaschi, B., S. Castano, S. D. C. d. Vermercati, S. Montanari and M. Vicini (1998). An Intelligent Approach to Information Integration. First International Conference on Formal Ontology in Information Systems, Trento, Italy, IOS Press, Amsterdam, The Netherlands. pp. 253-268.

Berners-Lee, T. (2001). Keynote presentation on web services and the future of the web. Software Development Expo 2001 Visionary Keynote, http://www.technetcast.com/tnc_play_stream.html?stream_id=616.

Berners-Lee, T. and M. Fischetti (1999). Weaving the Web, The original design and ultimate destiny of the World Wide Web, Harper.

Bressan, S., K. Fynn, C. Goh, M. Jakobisiak, K. Hussein, H. Kon, L. T., S. Madnick, T. Pena, J. Qu, A. Shum and M. Siegel (1997). The COntext INterchange Mediator Prototype. ACM SIGMOD International Conference on Management of Data, Tucson, Arizona. pp. 525-527.

Calvanese, D., G. D. Giacomo, M. Lenzerini, D. Nardi and R. Rosati (1998). Description logic framework for information integration. Proceedings of the 6th Interrnational Conference on the Principles of Knowledge Representation and Reasoning (KR-98). S. C. Shapiro. San Francisco, California, Morgan Kaufmann**:** 2-13.

Cardoso, J., J. Miller, A. Sheth and J. Arnold (2002). "Modeling Quality of Service for Workflows and Web Service Processes." The VLDB Jounal(submitted in May 2002).

Cardoso, J., A. Sheth and J. Miller (2002). Workflow Quality of Service. International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference (ICEIMT/IEMC'02), Valencia, Spain, Kluwer Publishers.

Christensen, E., F. Curbera, G. Meredith and S. Weerawarana (2001). W3C Web Services Description Language, http://www.w3c.org/TR/wsdl.

Fabio Casati, Ming-Chien Shan and D. Georgakopoulos (2001). "E-Services - Guest editorial." The VLDB Journal **10**(1): 1.

Fensel, D. and C. Bussler (2002). The Web Service Modeling Framework. Vrije Universiteit Amsterdam (VU) and Oracle Corporation, http://www.cs.vu.nl/~dieter/ftp/paper/wsmf.pdf.

Fensel, D. and M. Musen (2001). "The Semantic Web: A Brain for Humankind." IEEE Intelligent Systems **16**(2): 24-25.

Fox, C. (1992). Lexical analysis and stoplists. Information Retrieval: Data Structures & Algorithms. W. B. Frakes and R. Baeza-Yates. Englewood Cliffs, NJ, Prentice Hall**:** 102-130.

Garvin, D. (1988). Managing Quality: The Strategic and Competitive Edge. New York, Free Press.

González-Castillo, J., D. Trastour and C. Bartolini (2001). Description Logics for Matchmaking of Services. KI-2001 Workshop on Applications of Description Logics, Vienna, Austria.

Gruber, T. (1993). "A translation approach to portable ontology specifications." Knowledge Acquisition **5**(2): 199-220.

Guarino, N. (1998). Formal Ontology and Information Systems. Proceedings of Formal Ontology and Information Systems, Trento, Italy, IOS Press, Amsterdam. pp. 3-15.

Guarino, N., C. Masolo and G. Verete (1999). "OntoSeek: Content-Based Access to the Web." IEEE Intelligent Systems **14**(3): 70-80.

Hammer, J., D. McLeod and A. Soli (1994). An Intelligent System for Identifying and Integrating Non-Local Objects in Federated Database Systems. 27th International Conference on System Sciences, Honolulu, HI, Computer Society of IEEE. pp. 389-407.

Horrocks, I., F. v. Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness and L. A. Stein (2001). DAML+OIL, http://www.daml.org/2001/03/daml+oil-index.

Hull, D. A., J. O. Pedersen and H. Schutze (1996). Method combination for document filtering. Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Zurich, Switzerland, ACM Press, New York. pp. 279-287.

Jena (2002). The jena semantic web toolkit, http://www.hpl.hp.com/semweb/jena-top.html. Hewlett-Packard Company.

Kashyap, V. and A. Sheth (1993). "Schema Correspondences between Objects with Semantic Proximity," Department of Computer Science, Rutgers University, NJ, Technical Report DCS-TR-301.

Kashyap, V. and A. Sheth (1994). Semantics-based Information Brokering. Proceedings of the Third International Conference on Information and Knowledge Management (CIKM), Gaithersburg, Maryland.

Kashyap, V. and A. Sheth (1996). "Schematic and Semantic Similarities between Database Objects: A Context-based Approach." Very Large Data Bases (VLDB) Journal **5**(4): 276-304.

Kashyap, V. and A. Sheth (1998). Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies, Academic Press.

Kochut, K. J., A. P. Sheth and J. A. Miller (1999). "ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR," Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia, Athens, GA.

Krishnakumar, N. and A. Sheth (1995). "Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations." Distributed and Parallel Databases Journal **3**(2): 155-186.

Lee, H. L. and S. Whang (2001). "E-Business and Supply Chain Integration," Stanford University November 2001.

Lee, J., M. Kim and Y. Lee (1993). "Information Retrieval Based on Conceptual Distance in IS-A Hierarchies." Journal of Documentation **49**(2): 188-207.

Levenshtein, I. V. (1966). "Binary codes capable of correcting deletions, insertions, and reversals." Cybernetics and Control Theory **10**(8): 707-710.

Litwin, W. and A. Abdellatif (1986). "Multidatabase Interoperability." IEEE Computer **19**(12): 10-18.

Losee, R. M. (1988). "Parameter estimation for probabilistic document retrieval models." Journal of the American Society for Information Science **39**(1): 8-16.

Madhavan, J., P. A. Bernstein and E. Rahm (2001). Generic Schema Matching with Cupid. Proceedings of the 27th International Conferences on Very Large Databases, Roma, Italy. pp. 49-58.

McGuinness, D. (1998). Ontological Issues for Knowledge-Enhanced Search. Trento, Italy, IOS Press, Amsterdam, The Netherlands.

Mena, E., V. Kashyap, A. Sheth and A. Illarramendi (1996). OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. Conference on Cooperative Information Systems, Brussels, Belgium, IEEE Computer Society Press. pp. 14-25.

Miles, M. B. and A. M. Huberman (1994). Qualitative data analysis: an expanded sourcebook. Thousand Oaks, California, Sage Publications.

Miller, J. A., J. S. Cardoso and G. Silver (2002). Using Simulation to Facilitate Effective Workflow Adaptation. Proceedings of the 35th Annual Simulation Symposium (ANSS'02), San Diego, California. pp. 177-181.

Nelson, E. C. (1973). "A Statistical Basis for Software Reliability," TRW Software Series March.

Paolucci, M., T. Kawamura, T. R. Payne and K. Sycara (2002). Semantic Matching of Web Services Capabilities. Proceedings of the 1st International Semantic Web Conference (ISWC2002), Sardinia, Italia.

Parent, C. and S. Spaccapietra (1998). "Issues and Approaches of Database Integration." Communications of the ACM **41**(5): 166-178.

Porter, M. (1980). "An algorithm for suffix stripping." Program **14**(3): 130-137.

Richardson, R. and A. Smeaton (1995). "Using WordNet in a Knowledge-Based Approach to Information Retrieval," Dublin City University, School of Computer Applications, Dublin, Ireland, Technical Report CA-0395.

Rodríguez, A. and M. Egenhofer (2002). "Determining Semantic Similarity Among Entity Classes from Different Ontologies." IEEE Transactions on Knowledge and Data Engineering (in press).

Rommel, G. (1995). Simplicity wins: how Germany's mid-sized industrial companies succeed. Boston, Mass, Harvard Business School Press.

Salton, G. (1988). Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer. Massachusetts, Addison-Wesley.

Shegalov, G., M. Gillmann and G. Weikum (2001). "XML-enabled workflow management for e-services across heterogeneous platforms." The VLDB Joumal **10**(1): 91-103.

Sheth, A. and V. Kashyap (1992). So Far (Schematically) yet So Close (Semantically). Proceedings of the MT DS-5 Conference on Semantics of Interoperable Database Systems, Lorne, Australia, Elsvier Publishers.

Sheth, A. P., W. v. d. Aalst and I. B. Arpinar (1999). "Processes Driving the Networked Economy." IEEE Concurrency **7**(3): 18-31.

Sheth, A. P. and J. A. Larson (1990). "Federated database systems for managing distributed, heterogeneous, and autonomous databases." ACM Computing Surveys **20**(3): 183-236.

Song, M. (2001). RepoX: A Repository for Workflow Designs and Specifications. M.Sc. Department of Computer Science, University of Georgia, Athens.

Stalk, G. and T. M. Hout (1990). Competing against time: how timebased competition is reshaping global markets. New York, Free Press.

Sycara, K., M. Klusch, S. Widoff and J. Lu (1999). Dynamic Service Matchmaking Among Agents in Open Information Environments. SIGMOD Record. A. Ouksel and A. Sheth**:** 47-53.

Sycara, K., J. Lu and M. Klusch (1998). "Interoperability among Heterogeneous Software Agents on the Internet," The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA October 1998, pp. 35.

Thatte, S. (2001). XLANG: Web Services for Business Process Design. Microsoft, Inc., http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

Tidwell, D. (2000). Web services - the Web's next revolution. IBM, http://www-106.ibm.com/developerworks/webservices/.

Tversky, A. (1977). "Features of Similarity." Psychological Review **84**(4): 327-352.

Tversky, A. and D. Kahneman (1974). "Judgement under uncertainty: Heuristics and biases." Science **185**: 1124-1131.

Ulrich, F. (2001). "The Concept of Virtual Web Organisations." Electronic Journal of Organizational Virtualness: 43-64.

Uschold, M. and M. Gruninger (1996). "Ontologies: Principles, methods and applications." Knowledge Engineering Review **11**(2): 93-155.

Woelk, D., P. Cannata, M. Huhns, W. Shen and C. Tomlinson (1993). Using Carnot for enterprise information integration. Second International Conference on Parallel and Distributed Information Systems. pp. 133-136.

XMLSchema (2001). XML Schema Part 2: Datatypes. W3C Recommendation 02 May 2001, http://www.w3.org/TR/xmlschema-2/.

Zamora, E., J. Pollock and A. Zamora (1981). "The Use of Trigram Analysis for Spelling Error Detection." Information Processing and Management **17**(6): 305-316.