Ontology-Driven Web Services Composition Platform

I. Budak Arpinar, Ruoyan Zhang, Boanerges Aleman-Meza, and Angela Maduko

Large Scale Distributed Information Systems (LSDIS) Lab,
Computer Science Department, University of Georgia,
Athens, GA 30602-7404, USA
{budak,boanerg,ruoyan,maduko}@cs.uga.edu

Abstract. Discovering and composing individual Web Services into more complex yet new and more useful Web Processes is an important challenge. In this paper, we present three techniques for (semi) automatically composing Web Services into Web Processes by using their ontological descriptions and relationships to other services. In Interface-Matching Automatic composition technique, the possible compositions are obtained by checking semantic similarities between interfaces of individual services. Then these compositions are ranked considering their Quality of Services (QoS) and an optimum composition is selected. In Human-Assisted composition the user selects a service from a ranked list at certain stages. We also address automatic compositions in a Peer-to-Peer network.

Keywords: Web Services, Discovery, Composition, Ontology

1. Introduction

In recent years, a large number of Web Services (WSs) have emerged with the rapid development of the Internet. The Web is now evolving into a distributed device of computation from a collection of information resources [Fensel02a]. Furthermore, the need for composing existing WSs into more complex services is also increasing, mainly because new and more useful solutions can be achieved. In general, this is a result of complex and increasing user demands and inability of a single WS to achieve a user's goals by itself. For example, a traveler who wants to make a hotel reservation and find a French restaurant less than three miles from the hotel may either utilize some services that s/he already knows or try to find the services by looking them up in a keyword-based search engine (e.g., expedia.com or google.com) or in a WSs registry (e.g., a UDDI (Universal Description, Discovery and Integration) registry). Also, composition of discovered services and enabling data-flow among them (e.g., the hotel address is needed as input to a restaurant locator service) are usually done manually, which is highly inconvenient, especially for very complex compositions.

The problem lies with the fundamental abstractions used to model WSs and methods to compose these services using these abstractions. In more complex examples of scientific data exploration through service compositions, even tens or hundreds of data collection services can be involved in a composition (e.g., a search involving gene banks). In that case an automatic composition can help in reducing query formulation and execution time enormously.

In general, there are four different dimensions for a service composition (Figure 1): (i) degree of user involvement in a composition specification, (ii) whether the composition is based on templates, (iii) dynamicity (i.e., adaptation) of the composition, and again, (iv) degree of user involvement in the adaptation of the composition [Cardoso03 & Srivastava03]. In the first dimension, a composition can be defined fully by a user by including control and data-flow information besides the individual services making the composite service. In contrast, in an automatic composition, the user is not involved but instead the application generates control and data-flow. This is very challenging due to the difficulty of mapping user needs to a collection of correlated services where their interim outputs can satisfy one another's input requirements and final deliverable meets the user demands. Besides, in both user-defined and automatic composition techniques, either actual service instances or service templates can be used. In the latter, the individual service instances are searched and integrated automatically at execution time for a given plan [Chandrasekaran03]. In a dynamic composition, the composition itself can be adapted mainly because of Quality of Services (QoS) requirements at run-time. Also, a composition may not be defined at design-time but can be assembled dynamically at execution time. Finally, some hybrid methods such as semi-automatic compositions and semi-automatic adaptations are also possible.

This work aims for reducing the complexity and time needed to generate, and execute a composition and improve its efficiency by selecting the best possible services available using automatic Interface–Matching Automatic (IMA) and Human-Assisted (HA) composition techniques. IMA composition has no predefined template and the user is not involved in the composition specification. In contrast, the user is greatly involved in the HA composition specification and adaptation based on predefined templates.

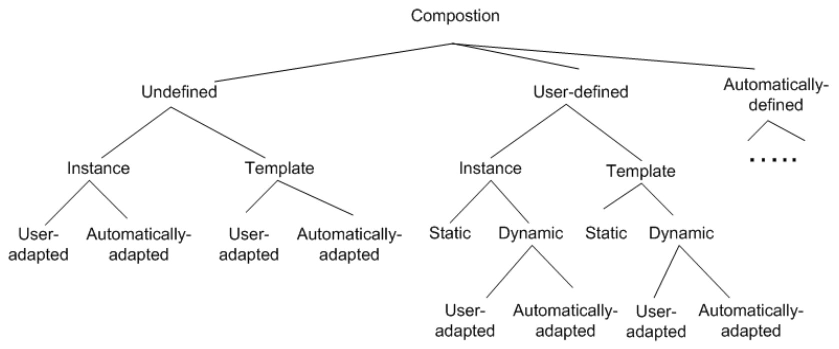

Figure 1: A Classification of Web Service Composition Techniques

1.1 Contributions

We developed a collection of ontology-driven Web services composition techniques. In IMA composition technique, the possible compositions are obtained by checking semantic similarities between interfaces of individual services with neither any predefined template nor user involvement in specification and adaptation. An optimum composition which can best satisfy the user's needs considering quality (or other attributes that the user may be concerned about) is selected. However, our experiments show that without functionality constraints, IMA technique is more appropriate for information-retrieval services (i.e., not world-altering services). That is, services that always return relatively simple results based on the user-supplied inputs [Zhang03 & Arpinar04]. This is mainly due to the fact that Web services with the same interface could perform different functions. There are also many earlier efforts in composition of software components using pipe-filter methods [Mao01]. Our contribution is that we developed a shortest-path algorithm to address a more complicated problem in which the services might have multiple I/O (input and output) parameters. This technique also takes Quality of Service (QoS) of WSs into account to find a best quality composition (e.g., with minimum cost) as described later. Furthermore, using the ontological techniques, we can successfully compose services even when their interfaces are not exactly matched syntactically.

However, complex services can not be located precisely and composed sequentially merely considering the interfaces. For example, flight booking and hotel reservation services, where the user's request is based on not only the inputs and outputs, but also on the QoS (e.g., service quality rate in DAML-S [DAML-S Coalition02]), functionality, service name, geographic regions, etc. As the number of various services increases, the task of selection of an adequate service quickly becomes tedious. Therefore, new techniques are needed to help users in finding, filtering, and composing these services.

At the beginning of service composition, a user may have a vague idea of the composite service that s/he is seeking. Another possibility is that an available composite template may not satisfy the user, who wants to add or remove some tasks in the original plan if necessary. In addition, service selection and composition are not only determined by the constraints on service description (e.g., interface), functionality properties and the service output values to be returned at run-time need to be considered. For example, consider a *trip planning* service composition example. A user often prefers a flight service that provides the cheapest flight ticket, which, however, could only be known after comparing the output ticket prices of a set of services. Furthermore, even the type of transportation service in the composition could be determined by the dynamic information provided by other services, such as parking fee or taxi rate if s/he is considering traveling by either flight or rental car. Thus depending on requirements on service descriptions and dynamic information supplied by the services, a customization and dynamic composition with more human involvement may be needed for many applications. A preliminary technique to have a run-time control over the composition is the introduction of an XOR-split in the control-flow to filter out undesired services (e.g., in price selection). An XOR-split is used to select only one of the several execution branches, which satisfies an associated condition.

The Human-Assisted (HA) composition technique targets applications which require frequent human-intervention for an acceptable composition. The goal of this technique is to build a composition pathway (i.e., map) incrementally in a customized way. This is a complementary technique to IMA that guides the users for selecting proper service instances described in DAML-S (version 0.9), and allows users to create a customized composite service plan.

In this paper, we intend to address the following issues:

1. Exploiting Semantic Web and Web Service Ontologies to bridge the concept gaps in interface parameters and other parts of descriptions of services. Basically, there are four types of semantics for Web Services: (i) data/information semantics, (ii) functional/operational semantics, (iii) execution semantics, and (iv) QoS semantics [Sheth03]. Our framework mainly uses the former two semantics and intends to support users in the selection of service classes and instances by matching service interfaces and attributes.
2. Ranking and filtering of composable services (a notion capturing if two services are interoperable, i.e., if they have complementary I/O interfaces and functionalities) at certain intervention points by the user. This also considers a semantic user profile and output values for filtering large numbers of service instances.

3. Automatically adjusting a composite service plan by removing the non-relevant services or adding the services suggested by the system if their I/O matches semantically.

Currently, descriptions of many WSs are contained in a single central registry, such as UDDI. Using emergent Peer-to-Peer (P2P) computing techniques, this registry can be moved from its centralized nature to a distributed one. With decentralization, problems of availability, reliability and scalability can be addressed. This, for example, can enable service providers to choose any particular registry in which their services would be listed. This might be beneficial in situations where competitors do not want to be listed in the same registry. Thus, we also briefly present a P2P Automatic (PPA) composition technique.

The rest of the paper is organized as follows: In Section 2, we review the related research. Section 3 briefly describes the system architecture. IMA, HA, and PPA composition techniques are described in Sections 4, 5, and 6 respectively. Section 7 concludes the paper, and outlines the future work.

2. Related Work

This work aims to generate a composite service plan out of semantically described existing services. The WS composition is related to many efforts. These include the WS specification, discovery, composition and execution techniques.

2.1 Web Service Description

The service specification methods help software systems to capture capabilities of WSs. In general, these specification methods are based on either industry-oriented standardization efforts, or academia-oriented WS ontologies. UDDI and Web Services Description Language (WSDL) are current industry standards developed for e-commerce. The services are described according to an XML schema which is defined in the UDDI specification and registered by the service providers along with keywords for their categorizations. Therefore, UDDI does not provide a semantic search rather it depends on a predefined categorization of WSs through keywords. In complimentary roles, WSDL and Simple Object Access Protocol (SOAP) describe WSs as a set of endpoints (or ports) operating on messages, and a protocol for exchange of these messages between the services, respectively. Also, some industry standards have been emerging to represent data and control-flow, and transactional properties among a collection of services. Business Process Modeling Language, XLANG, Web Services Flow Language (WSFL), and Business Process Execution Language for Web Services (BPEL4WS) can be mentioned in this category.

The semantic approach for WS specifications includes Web Service Modeling Framework (WSMF) in which an ontology provides the terminology used [Fensel02b]. DAML-S (recently evolved to OWL-S) specifies three main components for each service: service-profile, process-model and service-grounding. A service profile is the core element of a DAML-S specification, and involves semantic descriptions of service interfaces and functions [DAML-S Coalition02]. The process-model provides information about how the service works, and the service grounding describes how an agent can access the service. Other techniques have attempted to add semantics to existing services by providing mappings between WSDL, UDDI definitions and domain ontologies (e.g., METEOR-S [Sivashanmugam03a, Patil04a]).

In this work, we primarily focus on the collection of inputs, and outputs for composition and leaving pre-condition, and effect (i.e., post-condition) oriented composition as a future work.

2.2 Semantic Web Service Discovery and Composition

Semantic WS discovery related work includes [Cardoso03], which describes how to evaluate a degree of similarity between a service template and an actual service instance by measuring the syntactic, operational, and semantic similarity. Among the state-of-the-art discovery systems, the project DReggie adds reasoning modules to carry out the semantic matching process for discovering DAML described Web Services [Chakraborty01]. Unlike other discovery approaches on the basis of WS interfaces, [Klein01] explored ways to search services according to the functionality requirements, and proposed Process Query Language (PQL) to search process models from a *process ontology*. [McIlraith01, McIlraith02] presented a method to compose WSs by applying logical inferencing techniques on pre-defined plan templates. Their technique focuses on the process-centric description of service as actions that are applicable in states. Some earlier efforts also emphasized the need for semantic representations of state, actions, and goals for composing services [Srivastava03].

The main concept behind service composition is not new in Computer Science. Earlier, software composition techniques aimed to find a good combination of components that responds to the client specific requirements by matching requested properties with provided properties [Zaremski97]. One approach for finding a suitable composition is to delegate the responsibility for solving certain requirements posed on a component to other components after fulfilling it partially [Sora01]. Similarly, our interface-matching mechanism (IMA) propagates requirements (that are set of a user's expected outputs) to corresponding WSs in an incremental way. [Mao01] proposes a composition path, which is a sequence of operators that computes data, and connectors that provide data transport between operators. The search for possible operators to construct a sequence is based on the shortest-path algorithm on the graph of operator space. However, [Mao01] only considered two kinds of services – operator and connector with one input and one output parameter (which is the simplest case for a service composition) and did not take semantics into the account.

In the instance composition category, SWORD uses a rule-based expert system to determine if a plan of composite service can be built out of existing services [Ponnekanti02]. It mainly focuses on the composition of information provider services (i.e., not world-altering services), and (like [Mao01]) it does not address the input and output mismatch problem. In our approach, services can have more than one input and output parameters, and their interfaces may not match syntactically. We address semantic matching of parameters in the different composition methods proposed. Furthermore, some related work has identified types of semantic matchmaking [Palucci02] as well as partial matching [Constantinescu03].

2.3 Interactive and Adaptive Composition

At present, academic approaches have been proposed to tackle the problems for personalization and filtering of WSs based on templates. An example is a trip planner, which is declared as a state chart, and the resulting composite services are executed by replacing the roles in the chart by selected individual services [Benatallah02]. Template-based composition techniques are also used in [Narayanan02], and the ICARIS project [Tosic01].

Goal-oriented inferring and planning are used for services composition in the semantic Web community. However, none of the approaches has developed a satisfactory planning solution to the service composition so far [Srivastava03].

In [Sirin03], users select and filter the services by using a similar matchmaking algorithm. In [Balke03], the services are selected by using the hard and soft constraint standards in a personalized composition.

[Ambite03] designs a constraint reasoning network to compute any user's input change and produce the corresponding outputs to optimize the schedules for the trip based on the AI constraints reasoning technology. For a relatively fixed template, such as trip planner, generating a predefined constraints network might be feasible in practice. For less widely used services, it may not be possible to have such a priori constraints network. Our process ontology which is explained later has a similar feature with the constraints reasoning network except the latter only links services or functions to produce the expected results while in the process ontology all services would be connected only if their inputs and outputs are matched.

When the composite service plan is generated, a verification of the service logic is crucial for a successful execution. [Cheng02] presents an algorithm that checks the validity of the execution of services.

Also our earlier work has classified possible composition system architectures into three categories in the context of inter-organizational business processes, namely process portal, process vortex and dynamically trading processes [Sheth99].

2.4 METEOR -S Framework

The METEOR-S project at LSDIS at University of Georgia, has studied the use of emerging WSs and semantic Web technologies and research, to develop semantic WS and process specification, WS discovery, and process composition [Patil04b, Cardoso02, Sivashanmugam03b].

MWSAF (METER-S Web Service Annotation Framework) was designed to mark up WS descriptions with ontologies and develop algorithms to match and annotate WSDL files with relevant ontologies [Patil04a]. The METEOR-S Web Services Discovery Infrastructure (MWSDI) provides a scalable infrastructure for semantic publication and discovery of WSs [Verma04]. A specialized ontology called *Registries Ontology* maintains the relationship between all the domain and associates registries to them. Additionally, an algorithm has been developed to find the WSs with proper interfaces and operational mechanisms for workflow generation [Cardoso02, Cardoso03].

As part of the METEOR-S project, the MWSCF (METER-S Web Service Composition Framework) platform specifies an activity by means of a semantic activity template then weights the overall ranking of services on the two dimensions: semantic matching and QoS criteria matching [Sivashanmugam03]. Our work has benefited from the METEOR-S techniques that are mentioned. For example, the similarity algorithm described in Section 4 is partially based on the algorithm in [Cardoso02].

2.5 Peer-to-Peer Web Services Composition

A system for declaratively composing and executing WSs dynamically in a Peer-to-Peer (P2P) network is presented in [Benatallah02]. The concept of a service community is used to provide an abstraction from actual service providers so that any WS with the functionality described by the service community can be a member of that community, regardless of its service domain. However in this work, we focus on how semantic WSs hosted by nodes in a P2P network can be grouped in order to reduce the search space during discovery and composition of WSs. Host peers of WSs are automatically grouped into service communities, where a service community refers to a group of WSs in a given domain, determined by a globally shared domain ontology.

[Schlosser02] presents a P2P infrastructure for Semantic WSs discovery (without addressing services composition). Peers are grouped into ontological concept clusters in a Cayley graph based structure, with the clusters organized into hyper-cubes, so that queries can be sent to only those peers that are potentially able to answer them. When a query is routed to a concept cluster, the query is broadcasted to all peers within the cluster in order to discover a peer that can answer the query. On the contrary, we have the concept of *master peers* for each service community. Since the master peer has information about all the peers within a community, it can determine which peers can answer the query, thereby minimizing the amount of communication amongst peers in that community. Furthermore, the infrastructure of [Schlosser02] does not take into consideration quality of service metrics. In our approach, the master peer for each community can select a peer from amongst candidate peers for a particular query based on its quality of service.

Other works such as [Hoschek02, Schmidt03, Banaei-Kashani04] focus on web services discovery on a P2P paradigm. They do not however consider how the WSs can be dynamically composed.

3. An Overview of System Architecture

3.1 Specification of Semantic Web Services and Queries

A semantic WS is a unit of composition that can be deployed independently, and may be subject to composition by a third party on the Web. At the same time, its interface, its process specification (i.e., its functionality) and its relations to other services are defined, and advertised in a machine-processable form so it can be automatically discovered, composed, and invoked in new complex WSs. The emerging semantic Web makes it possible to specify semantics of a domain such as the terms and concepts of interest, their meanings, relationships between them and the characteristics of the domain through an ontology. In this work, we use DAML-S (version 0.9) WS ontology. A service profile is the core element of a DAML-S specification, and it involves semantic descriptions of service interfaces and functions. In this work, we primarily focus on the collection of inputs, and outputs for composition.

A composite service query can be represented in a very similar way as a service description in DAML-S. Like DAML-S template of services, the query profile includes the description of the composite service and the interface of the expected composite service, in which we define the output parameters, output constraints, input parameters, and input constraints. The output constraint specifies the requirements on the outputs by the user, such as the properties of the output parameter. For example, the user can define price properties, currency, etc. using a price ontology.

The second part of the query is about the functionality of the composite service (part of our future research). The user can partially specify how the composite service works and what kind of individual services would be expected. For example, a restaurant owner may want to find matching wines for certain meals in her restaurant and learn the prices of these wines. To formulate a query s/he can specify a seafood type for a Food-Wine-Matching service and expect the names and prices of matching wines.

3.2 System Architecture

The system architecture (Figure 2) involves three components (i) composer component, (ii) ontology and service storage component and (iii) extraction component. The storage component hosts the user profile, WS, process and domain ontologies. User profile records the history of each user's usage of WSs. The service instances can be ranked based on their usage frequency (i.e., popularity).

Ontology component includes domain ontologies (in OWL) that are specialized for description of parameters of the services. For example, an ontology for food and drink related WSs specifies the sub-class and super-class relationships for the relevant entities and properties [Zhang04]. In this ontology, *Wine* is defined as a subclass of the *Alcohol* that is the subclass of *Drink*. The properties of *Wine* include *wine name, vintage, merchant location* and *price*, etc. The user can specify the properties of *Wine* in a query and limit the range of the properties.

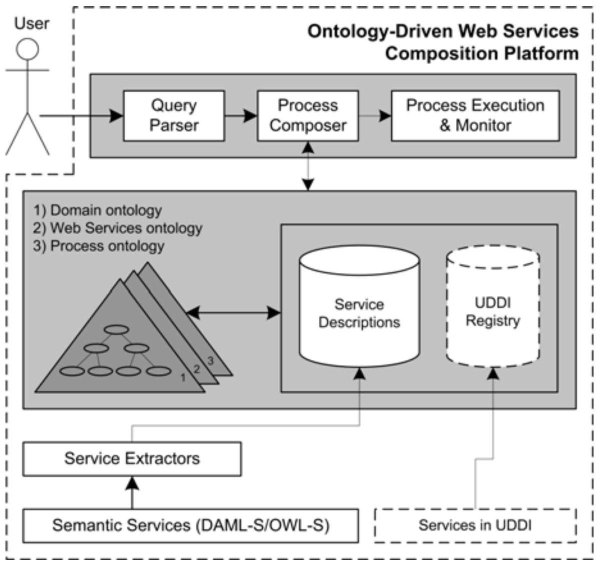

Figure 2: An Overview of System Architecture

Like domain ontologies, a WS ontology describes service hierarchies and a subclass of a service inherits the properties and functionality of its super-class service and extends it with its own attributes [Zhang04]. A *process ontology* involves a collection of services that are connected each other if their interfaces are matched semantically and built for improving the efficiency of composition algorithm (Figure 3).

The query parser parses the query and sends the query object to the process composer, which searches for a sequence of services by navigating WS and process ontologies and returns a composite service with the optimal graph. It also generates the data-flow plan of the composite service and sends the plan for process execution and monitoring.

When a service provider sends a registration request, a service extractor extracts service name (including its ID), text description, instance of relationship, input/output information, etc. from the service profile and stores them in a services database. The process ontology is also updated by connecting the I/O parameters of the new service to other compatible service parameters. This helps in reducing the complexity of searching for services in an automatic composition as mentioned earlier. If the services are described in UDDI schema, their profiles would be sent directly into a UDDI registry.

4. Interface-Matching Automatic (IMA) Composition

IMA composition technique aims for generation of complex WS compositions automatically. This requires capturing user's goals (i.e., expected outcomes), and constraints, and matching them with the best possible composition of existing services. Therefore, inputs and outputs of the composite service should match the user-supplied inputs, and expected outputs, respectively. Furthermore, the individual services placed earlier in the composition should supply appropriate outputs to the following services in an orchestrated way similar to an assembly line (i.e., pipe-and-filter) in a factory so they can accomplish the user's goals.

In IMA, we navigate the process ontology to find the sequences starting from the user's input parameters and go forward by chaining services until they deliver the user's expected output parameters. The composition terminates when a set of WSs that matches all expected output parameters is found, or the system fails to generate such a composition of services.

The goal of this technique is to find a composition that produces the desired outputs within shortest execution time and better data-flow (i.e., better matching of input and output parameters). If the service ontologies are complex and the number of services is large this can be a challenging task. The composition starts from the service that needs one or more of the input parameters given by the user. If this WS does not produce all of the expected outputs, more WSs need to be found to provide the expected outputs. This process continues until a sequence of WSs producing the expected composition outputs from the user's inputs is found.

Figure 3 shows a sample process ontology involving relations representing input and output parameter matching. Nodes represent services and edges connect services if the output of a service can be "feed-into" the input of a service. Edges shown with dashed lines connect parameters that do not match exactly yet are semantically equivalent. In the figure, different service outputs can feed into other service inputs. For example, Service 6 requires two input parameters, one of which can be provided by either Service 1 or Service 3 and the other comes from Service 4.

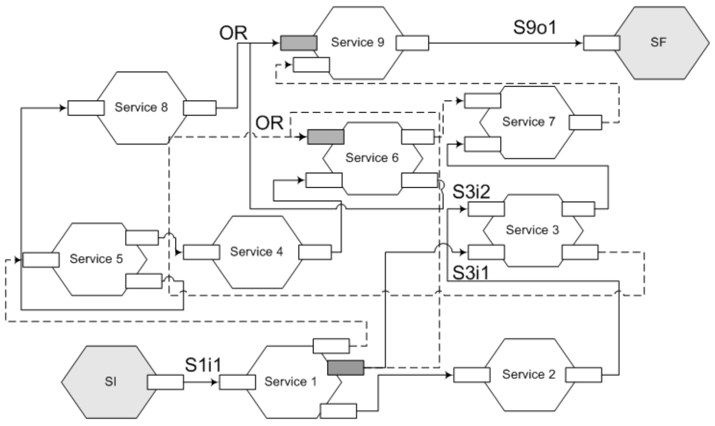

Figure 3: A *Process Ontology* Example

In an example scenario, the user provides input parameter S1i1 and expects the output S9o1 *as* indicated in the graph. The composition goal is to find a shortest sequence of services from Service 1 to Service 9. In this graph the source node SI and SF represent the virtual initial and final services respectively, which are added for computing convenience. The weight of every edge is calculated using a function of quality rate and semantic similarity value. To assign weights using quality five generic QoS criteria can be used: (1) execution prices, (2) execution duration, (3) reputation, (4) reliability, and (5) availability [Zeng03].

The trade-off formula weighting quality criteria versus similarity is defined by the following formula, with a user adjustable parameter λ:

$$W = (1-\lambda) * \text{quality rate} + (\lambda) * \text{similarity value}.$$

We considered four cases to check similarity (i.e., matching) of an output and input parameter from the same ontology: (1) if they are same, their similarity is maximal and weight of the corresponding edge is the smallest. (2) If the output parameter of the former service is subsumed by the input parameter of the succeeding service, this is the second best matching level. The similarity value depends on their distance in the ontology. (3) If the output parameter of the former service subsumes the input parameters of the succeeding service, the properties of the parameters could be partially satisfied. (4) When two parameters have no subsumption relation or they are from different ontologies, the similarity value can be obtained by using Tversky's feature-based similarity model [Cardoso02], which is based on the idea that common features increase the similarity of two concepts, while feature difference decreases the similarity.

The composition technique aims to find an optimal composition of services considering QoS and semantic matching of parameters as illustrated in Algorithm 1. We modified Bellman-Ford shortest-path dynamic programming algorithm to find the shortest sequence from initial stage at node SI to the termination node SF. In our graph representation some services need more than two incoming edges as input parameters. Therefore, we not only record distance for every node, but also we trace the distance of every path at every node. When all the required input parameters are available, a service can be executed. Therefore, the distance of every node is determined by the maximum value of distances of all the input parameters. For example, Service 3 must have two incoming edges and therefore a distance value of Service 3 is determined by the maximum of S3i1 and S3i2 because Service 3 can be executed only after both of these inputs are available (Figure 3). In a different case, when there is more than one incoming edge fitting for one input parameter of a service, such as output of either Service 1 or 3 satisfies input of Service 6, we choose the minimum distance of edges 3-6 and 1-6 as a distance associated with the input parameter of Service 6.

```
01. procedure compose(usersinputs, usersoutputs) returns Boolean
02. for times = 1 to N // in worst case, the algorithm needs to update distance of each
                            service N times (N is the number of services)
03.     for i = 1 to N // check each service
04.         for $i_{in}$ = 1 to $N_{in}$ // $N_{in}$ is the number of inputs of service i
05.             for j = 1 to N // for each input of service i check each other
                                         service
06.                 for $j_{out}$ = 1 to $N_{out}$ // check each output of service j
07.                     Check the output $j_{out}$ of service j and the
                         input $i_{in}$ of service i to see whether they
                         are similar (connected); if similar find the
                         degree of their similarity
08.                 end
09              end
10.             Find the shortest distance to $i_{in}$ of service i (minimum of all
                available paths to $i_{in}$)
11.         end
12.         Find the shortest distance for service i (if outputs of service i need all
            inputs then its is the maximum distance of all inputs plus its quality)
13.     end
14.     If all required usersoutputs are obtained then return true
15. end
```

Algorithm 1: IMA Composition Algorithm

The complexity of Algorithm 1 is O($N^3.N_{in}.N_{out}$) where N is the number of available services and N_{in} and N_{out} are number of inputs and outputs per WS respectively. However this complexity can be improved by pre-processing WSs to gather similarity and distance information prior to WS composition.

Furthermore, obtaining quality rates of WSs is a challenging task. For this purpose we adopt a WS quality model similar to one proposed in [Sheng03] which is based on a set of quality criteria that are in inherent to WSs in general such as execution price, execution duration, reputation, reliability, and availability. The overall quality rate is an average of these individual rates. New criteria can be added to this quality model if needed. However our work does not propose any particular model for calculating these rates and mainly rely on service providers for advertising execution price, duration, and inquiring about availability of particular WSs. Calculating reliability and reputation rates usually need analyzing historical data and user feedback respectively. [Sheng03] provides more details about how to obtain these quality rates.

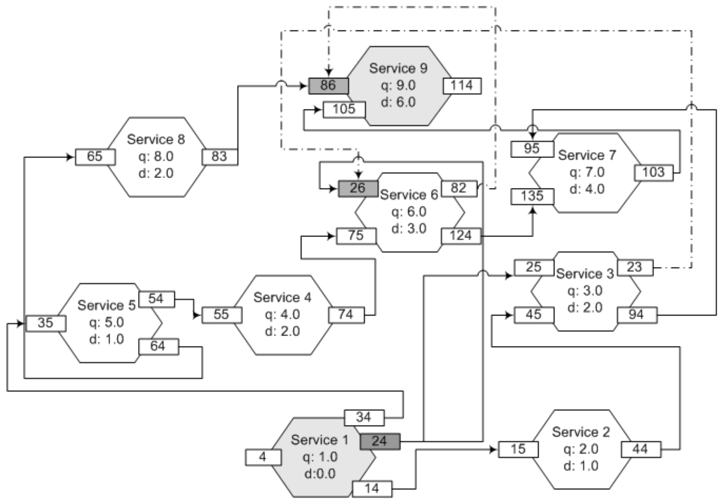

Figure 4: An IMA Composition Example (λ =0.3)

4.1 Automatic Composition Examples

Figure 4 illustrates a composition result with the given process ontology of Figure 3. ‘q’ is the quality rate of a service and ‘d’ is the distance from start service to this service. Note that the input and output parameters are assigned integer numbers. The smaller difference of two integers means the associated I/O parameters are more similar semantically. In this example, the input parameter for the composition is the input of Service 1 (4), the expected output is the output of Service 9 (114), and λ is 0.3. The shortest distance from Service 1 to 9 is 6.0. Note that Service 8 feeds one of the inputs of Service 9 rather than Service 6. And Service 6 selects the input from Service 1 instead of Service 3. The selected composition is illustrated with *solid* directional edges between the services.

If we change λ to 0.7 this means the quality rate has less weight in matching function. The composition with shortest distance is shown with solid lines in Figure 5. The shortest distance from Service 1 to 9 is 9.0. In this case, Service 6 is the input provider for Service 9 rather than Service 8.

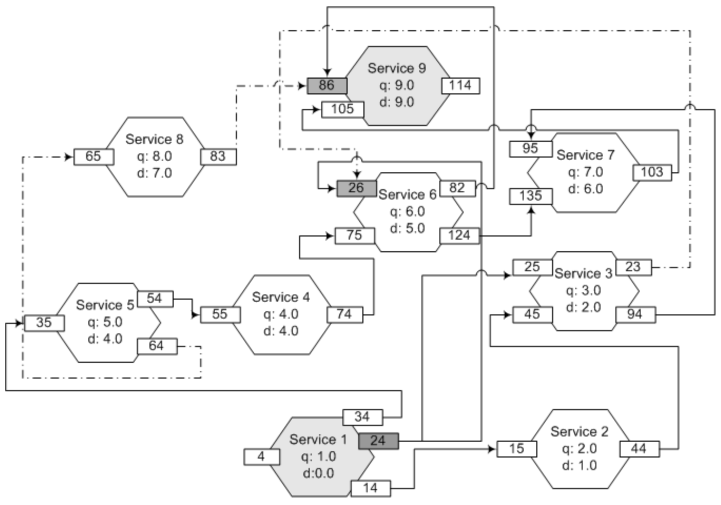

Figure 5: An IMA Composition Example (λ =0.7)

Figure 6 presents a practical example which we tested for the validity of proposed IMA composition technique. *Forster, Berry and Wine Answers* are three *Food-Wine Matching* services that provide the matching wine given food type or recipe. *Wine Searcher, Internet Wine* and *K. L. Wine* return the wine prices to the user given the wine name. The *Converter* service can convert a wine price in US dollars to Franc currency. *Recipe* service always presents a seasonal recipe if the user inputs the food name, such as beef.

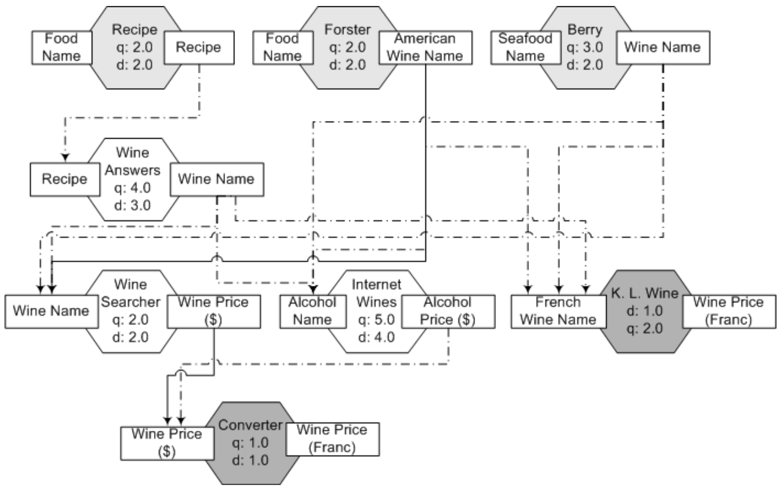

Figure 6: Food-Wine Matching Service Composition (λ =0.2)

Assume that the user inputs a seafood name and awaits the matching wine prices in Franc. All of three services accept such seafood as inputs. When λ is 0.2 and I/O similarity has a low priority, the shortest path is *Forster* → *Wine Searcher* → *Converter* (shown with solid line in Figure 6). The *Berry* matching service has the exact input parameter as the user's input, and *Forster* takes food as input, that is the super-class of the seafood. That is regarded as a second best matching in our approach.

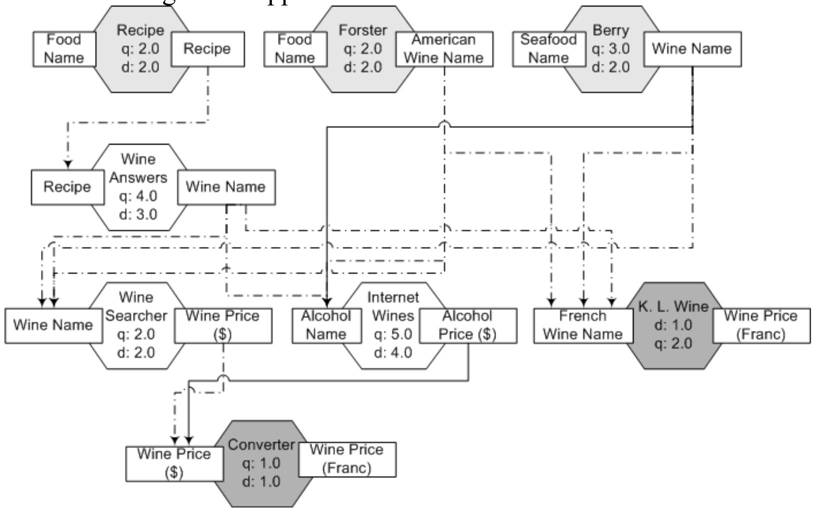

Figure 7: Food-Wine Matching Service Composition (λ =0.8)

When we increase λ to 0.8 that means we place more weight to similarity matching. The optimal sequence is shown in Figure 7. The shortest path is *Berry* → *Internet Wines* → *Converter*. The reason is that *Internet Wines* has a bad quality rate (5), but it has a good matching degree. Therefore, when λ is large and we pay more attention to the quality rate, *Internet Wines* is not a good choice. *K. L. Wine* is another service that can produce the expected result and it is adjacent to the food-matching services. The absence of this service is due to low similarity degree. The output of preceding service is *Wine* and *K. L. Wine* only accepts *French Wine* that is the subclass of the *Wine* and cannot meet the I/O requirement.

As illustrated in this section λ enables users to adjust relative importance of interface matching and QoS for a composition. Although we did not conduct extensive experiments for selecting good λ values in different situations we can provide some general guidelines here. We believe that a perfect interface matching is essential for successful enactment of a WS composition. Thus we don't advise using smaller values of λ (e.g., $\lambda < 0.5$). In situations where there is minimal information about QoS characteristics of WSs even greater values of λ (e.g., $0.8 \leq \lambda$) can be used.

5. Human-Assisted (HA) Composition

The goal of HAA composition is to help users in selecting appropriate WSs, and build a composition incrementally. The first step is to consider all inputs by semantically matching them with all WSs that take one or more of them as input. A list of these WSs is provided to the user with the WSs ranked based on their similarity matching score. The user can select WSs s/he considers best for the desired composition. Furthermore, to facilitate a better selection process, each listed WS includes a description of its functionality as well as the output(s) it produces. Then the system determines whether all output parameters of the desired composition are produced by the services selected so far. If that is the case, the composition is completed. Otherwise the interactive composition process continues with more stages, or can be terminated by the user. Figure 8 shows the first stage of a composition for a vacation trip arrangement, where the user has selected two WSs so far.

For the second stage, a new set of input parameters is generated by the system. This set includes the input parameters considered and also includes all outputs of the WSs selected by the user. The user is given the option to discard elements in this new set of input parameters that may no longer be needed. The user may also mark some of the new input as "optional". This helps in the ranking of the list of WSs that will be shown to the user in subsequent stages. The list of ranked results is grouped by input parameters to facilitate selection when the list is large. In the example of Figure 8, *Restaurant Style*, *Restaurant Place*, *Hotel Place*, and *Hotel Rate* are no longer considered as inputs in next stages. The inputs *Check-in* and *Check-out dates* and *Personal information* are still considered; outputs *Restaurant Location*, *Hotel Location*, and *Hotel Name* from the selected services are now considered as inputs in next stages; and output *Restaurant Name* from one of the selected services is no longer considered as input (unless stated otherwise by the user), because it satisfies an output parameter *Restaurant Name*. The smaller dashed box highlights the new set of inputs for the next stage, and in this way the composition problem has been reduced. Figure 8 also shows the second and final stage of the composition.

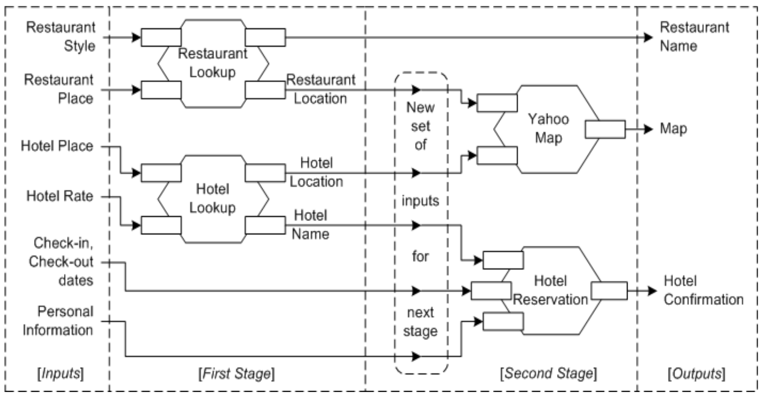

Figure 8: Stages of an Interactive Composition

6. Peer-To-Peer Automatic Composition

In this section, we present our techniques for automatic WS composition in a Peer-to-Peer (P2P) network, where the peers all share the same ontology.

6.1 Communities

In a P2P network, each peer can provide some WSs, where each WS serves a particular *domain*. Each peer is a member of at least one community. A *community* of peers refers to a group of peers that provide services for the same domain. The structure of the communities adheres to that of the ontology, so that where the ontology is hierarchical, the communities have a hierarchical structure. For example, if in the ontology there is a concept *motel* which is sub-concept of another concept *hotel,* then there would be *motel* and *hotel* *communities,* with the former being a *sub-community* of the latter. Each community has both a *master* peer and a backup *peer*. The *master* peer in each community maintains a list of the masters and backup peers of other communities, and backup peers have a replica of this list.

6.2 Evolution of the Network

Our P2P network evolves in two different dimensions (Figure 9): (1) the first dimension is based on the domain for which the services are provided; (2) the second dimension is based on input-output matching relationships amongst the peers' services. (The second dimension is not depicted in Figure 9). For the first dimension, each master peer maintains a list of all peers within its community with the services they provide as well as the input and output parameters they accept and generate, respectively. For the second dimension, peers become involved in a *predecessor-successor* relationships. The *successor* peer always takes note of this relationship for the progression of the composition process, as we shall see later in Section 6.3.

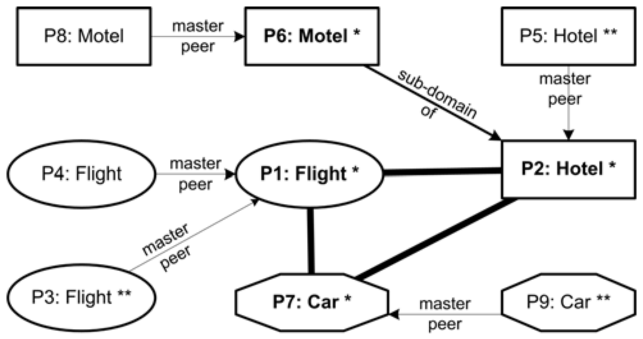

Figure 9: Ontology-driven P2P Network
(Different shapes indicate different domains; thick lines indicate communication lines between super peers (denoted by *, backup peers are denoted by **))

Consequently, the network is structured in such a way that peers with similar functionality are grouped together with all peers taking cognizance of data dependencies. The following explains how the network evolves in both dimensions (using the example in Figure 9).

(1) Initially there is only one peer (P1) in the network; when P2 contacts P1 to join the network, they determine based on their common ontology, what domains they provide services for, in this case "Flight" and

"Hotel" respectively. Since they are the first peers for these domains, they automatically become the master peers and record this in their lists. P3 contacts P2 to join the network, as before, based on the common ontology, P2 determines that P3 is in the "Flight" domain. Since P2 is itself a master of its domain it determines which peer to redirect P3 to (in this case P1). P3 then contacts P1 and records that P1 is its master. Since P3 is the only other peer in the community besides P1, it is automatically made the back up peer for this domain. This self-organizing network can also handle situations where a peer about to join the network contacts a non-master/back up peer as occurs when P5 joins the network, or where peers form different sub-communities within a community (e.g., P6 is a master for "Motel" sub-community of "Hotel" services).

(2) As each peer joins the network, its master queries other master peers of different communities to determine if there is any peer in their community for which the new peer's output matches any of their input or for which their output matches any of the new peer's input. In the first case, the new peer becomes a "predecessor" of any such peer and in the second case it becomes a "successor" of any such peer. If such a peer exists, the master peer in the community of the successor peer notifies it about this association. Thus, a directed graph with input/output compatibility is maintained.

6.3 Web Services Composition Technique

In this section, we describe our techniques for discovering and composing WSs in the P2P network as illustrated in Algorithm 2. We assume that each user query for a composite service specifies the expected functionality of the service as described in Section 3.1. When a peer (initiator peer) receives a request from a user for a particular service or service composition, if it is not the master of its community, it forwards the request to the master in its community. Then, the master of the initiator peer determines the candidate communities for the query and then relays the request to the master peers of these communities. These masters then determine which services in their community provide all or some of the expected outputs of the user, what inputs these services require and their host peers. If there are several candidate WSs (peers) for a particular service, one of them is chosen either randomly, or based on some quality of service criteria if any exists.

```
01.  procedure compose(usersinputs, usersoutputs) returns Boolean
02.  if initiatorpeer not domain master
```

```
03.         forward request to its domain master
04.  boolean reply = true
04.  for domainMasters = each domain master of candidate communities
06.         reply = reply & domainMaster.checkComposeable(userinputs,  useroutputs)
05.  if reply == true
06.         return true

01.  procedure checkComposeable(usersinputs,usersoutputs) returns Boolean
02.  if mycommunityoutputs == any of the usersoutputs
03.  boolean composeableViaMe = true
04.         aCandidatePeer = one peer producing any of the usersoutputs
                // with greatest QoS
05.         composeableViaMe = checkComposeable(aCandidatePeer, aMaster
            Peer.usersInputs) & composeableViaMe;
06.         if composeableViaMe is false and all candidate peers have been tried
07.                return false  else return true

01.  procedure checkComposeable(peer, usersinputs) returns Boolean
02.  if userinputs contains all peers inputs
03.         return true
04.  else if peer does not have dependencies on all needed inputs  // successor peer
05.         return false
06.  else
07.         boolean canCompose = true
08.         for predecessorPeer = 1 to # of my predecessor peers for the inputs I need
09.                canCompose = canCompose & checkComposeable(predecessorPeer,
                   usersinputs)
10.         return canCompose
```

Algorithm 2: P2P Composition Algorithm

In general, composition of WSs is possible only when there are data dependencies amongst the individual services. In other words, the output(s) of some WSs has to match the input(s) of at least one WS. In our P2P network, these data dependencies have been captured by the predecessor-successor relationships amongst peers as mentioned earlier. Discovery of peers that can participate in the composition amounts to traversing these predecessor-successor relationships, beginning with the peer(s) producing the user's outputs, up to those accepting the inputs (provided by the user) required for the composition. This is possible because as we mentioned in section 6.2, each successor peer knows its predecessor peer(s). For example, in Figure 10 the user sends the request with input *a*,*b*,*c*,*d*,*e*,*f* and output *x*,*y*,*z* to Peer 13 (P13). P13 forwards this request to its master P7, which then determines based on the ontology to forward this request to P1 and P2. These master peers then look up in their table which peers within their community provide some or all of the expected outputs. P2 determines that P10 and P5 are candidates for the composition. Since the input *g* of P10 does not match any of the inputs provided by the user, P10 contacts P4, its predecessor peer. P4 then matches its inputs with those provided by the user. In this case, there is a match so this predecessor-successor traversal ends. Also, P7 determines that P15 can participate in the composition. The actual composition then starts with P4, P15, P5 and then P10.

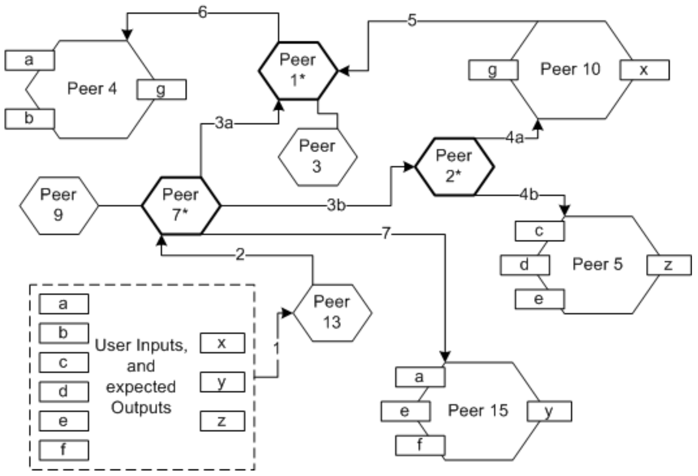

Figure 10: Example of P2P-based Composition (super-peers are denoted by *)

6.4 Evaluation

For proof of concept, we have simulated our P2P composition technique. A peer is represented by a Java object, each identified by a *peerId.* We randomly assigned WSs from different service domains to peers, and experimented with different numbers of peers, each time, observing the time taken for the evolution of the network. For each number of peers tested, we supplied two types of queries for composite services, namely composeable and non-composeable queries. We observed the time taken for the discovery of the services for the composition for the first type of queries. For the second type of queries, we observed the amount of time it took to report that the requested service could not be composed. The simulation was carried out using an Intel Pentium IV processor with 2.66GHz processor speed and 512MB RAM.

As shown in Table 1 and Figure 11, as the number of peers increased, the time taken for the network evolution time naturally increased. However, the times taken to run the composeable queries were approximately the same. This is so because all peers keep an index of which peers it depends on for any input. As such, irrespective of the number of peers in the network, it should take approximately the same amount of time to determine if the requested service can be composed, since only the peers that can potentially answer the query are contacted. On the other hand, for the non-composeable queries, the amount of time taken to discover that the requested service can not be composed increased proportionally with the number of peers. This is so because as the number of peers increased, the number of peers within a community also increased. Consequently, the number of candidate peers for each query also increased. In the first of the two types of queries (composeable queries), an early termination of the back-tracking to candidate peers is possible. For a candidate peer, an early termination of its predecessor/successor relationships is also possible. However, in the second type (non-composeable), the back-tracking does not terminate until all candidate peers have been contacted and all predecessor/successor relationships for all candidate peers have been traversed. Consequently, as the number of peers in the network increased, the latter type of query took more time than the former type.

Table 1: Simulation Results for P2P Network Evolution and WS Composition

		Elapsed Time (Secs)	
Number of Peers	Evolution Time(Secs)	Composeable	Not Composeable
100	0.06	0.01	0.01
500	0.33	0	0.03
1000	1.723	0.01	0.18
2000	9.534	0	1.632
5000	181.571	0.01	41.49

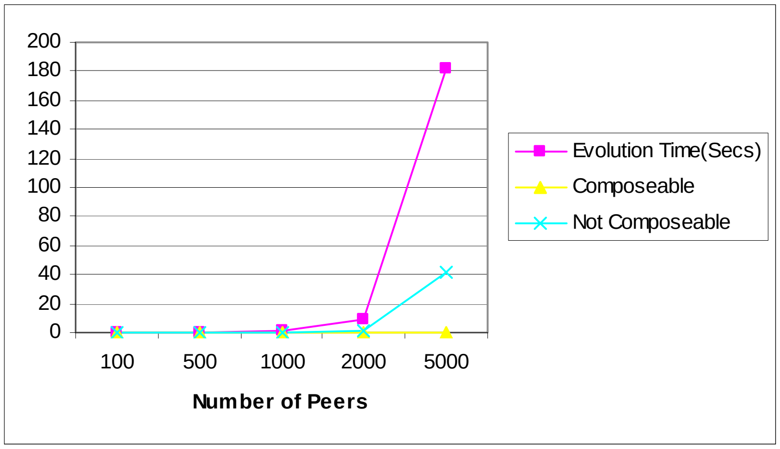

Figure 11: Simulation Results in Graph Format

7. Conclusions and Future Work

Today's search engines and knowledge discovery tools help users to locate relevant documents and assemble relevant knowledge for effective decision-making respectively. Similarly, users need new tools to help them discover and assemble WSs into processes for easier and better quality workflow executions given increasing number and complexity of WSs. In this paper, we illustrate three techniques for (semi) automatic composition of WSs by exploiting semantics provided by ontological description of inputs and outputs of the services. In Interface-Matching Automatic composition, the different services are put together to generate a set of expected outputs from the user. Our approach finds an optimal composition based on factors of semantic matching of inputs/outputs and QoS criteria (adjustable by users).

Human-Assisted composition follows the approach of iteratively composing the service by making use of filtering and ranking based on user-provided constraints. Factors such as geographic location, quality rate, cost, etc. can be taken into consideration at each stage besides the semantic matching of inputs/outputs.

The third technique presented deals with WSs composition in a P2P environment by utilizing master-peers of the network. The peers are (logically) organized in a dimension based on the domain(s) of the service(s) they provide. A second dimension keeps relationships of compatible semantic input/output matches among the peers' WSs regardless of the domain. This allows for a dependency (directed) graph organization that resembles that of our Interface-Matching technique.

For testing proposed techniques, collections of synthetically generated WSs in various domains such as travel planning, wine suggestion etc. are used. However real world applications of the proposed techniques are possible in these and other domains, such as bioinformatics to automatically compose WSs to integrate multiple genetics databases and applications and discover interactions among proteins [Kochut03].

Some interesting technical problems still lie ahead. For example, users may need to compose services based on their internal computations when their profiles may not convey adequate semantics to differentiate them. We plan to consider validation of the composition based on pre- and post-condition semantics.

References

[Ambite2003] J. Ambite, G. Barish, Craig A. Knoblock, M. Muslea, J. Oh, and S. Minton. Getting from Here to There: Interactive Planning and Agent Execution for Optimizing Travel. The Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI), Edmonton, Alberta, Canada, 2002.

[Arpinar04] I. B. Arpinar, R. Zhang, B. Aleman, and A. Maduko. Ontology-Driven Web Services Composition. IEEE E-Commerce Technology, July 6-9, 2004, San Diego, CA.

[Balke03] W. Balke, and M. Wagner. Towards Personalized Selection of Web Services. WWW 2003, May 20-24, 2003, Budapest, Hungry.

[Banaei-Kashani04] F. Banaei-Kashani, C. Chen, and C. Shahabi. WSPDS: Web Services Peer-to-peer Discovery Services. The 2004 International Symposium on Web Services and Applications, Las Vegas, NV.

[Benatallah02] B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. IEEE Intl. Conf. on Data Eng., San Jose, 2002.

[Cardoso02] J. Cardoso. Quality of Service and Semantic Composition of Workflows. Ph.D. Dissertation. Department of Computer Science, University of Georgia, Athens, GA, 2002.

[Cardoso03] J. Cardoso, and A. Sheth. Semantic e-Workflow Composition, Journal of Intel. Info. Sys., 2003.

[Chakraborty01] D. Charkraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: A smart Service Discovery Technique for E-Commerce Applications. 20th Symposium on Reliable Distributed Systems (SRDS). New Orleans. October, 2001.

[Chandra03] S. Chandrasekaran, J. Miller, G. Silver, I.B. Arpinar, and A. Sheth. Performance Analysis and Simulation of Composite Web Services. Electronic Markets: The Intl. Journal of Electronic Commerce and Business Media, 13(2), 2003.

[Cheng02] Z. Cheng, M. P. Singh and M. A. Vouk. Composition Constraints for Semantic Web Services. In Proceedings of the International Workshop Real World RDF and Semantic Web Applications, 2002.

[DAML-S Coali- A. Ankolenkar, M. Burstein, et. Al. DAML-S: Web Service Description03] tion for the Semantic Web. The First International Semantic Web Conference, Stanford, 2001.

[Fensel02a] D. Fensel, C. Bussler. Semantic Web Enabled Web Services. 2nd Annual Diffuse Conference, Brussels, Belgium, January 2002.

[Fensel02b] D. Fensel, C. Bussler, Y. Ding, and B. Omelayenko. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications, 1(2), 2002.

[Hoschek02] W. Hoschek. Peer to Peer Grid Databases for Web Services Discovery, Grid Computing: Making the Global Infrastructure a Reality" Ed(s): F. Berman, G. Fox, and T. Hey, Nov. 2002, Wiley.

[Klein01] M. Klein, and A. Bernstein. Searching for Services on the Semantic Web Using Process Ontologies. International Semantic Web Working Symposium, August 2001.

[Kochut03] K. Kochut, J. Arnold, A. Sheth, J. Miller, E. Kraemer, I. B. Arpinar, and J. Cardoso, IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions, International Journal of Distributed and Parallel Databases (DAPD), Special issue on Bioinformatics, , Volume 13, No. 1, January 2003.

[McIlraith01] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. IEEE Intel. Sys. March/April 2001.

[McIlraith02] S. MaIlraith, and T. C. Son. Adapting golog for composition of semantic Web services. In Proc. KRR, 482-493.

[Narayanan02] S. Narayanan, and S. A. Mcllraith. Simulation, Verification and Automated Composition of Web Services. 11th Intl. WWW Conference, Honolulu, 2002.

[Palucci02] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. The First Intl Semantic Web Conference, Sardinia (Italy), June, 2002.

[Patil04a] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web Service Annotation Framework, Proceeding of the World Wide Web Conference, July 2004.

[Patil04b] A. Patil. METEOR-S Web Service Annotation Framework. Master Thesis, Computer Science, University of Georgia, 2004.

[Ponnekanti02] S. R. Ponnekanti, and A. Fox. SWORD: A Developer Toolkit for Building Composite Web Services. 11th WWW Conference, Honolulu, 2002.

[Schlosser02] M. Schlosser, M. Sintek, S. Decker, and W. Neijdl. A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. 2nd IEEE Intl. Conf. on Peer-to-Peer Computing, 2002.

[Schmidt03] C. Schmidt and M. Parashar. A Peer-to-Peer Approach to Web Service Discovery, World Wide Web Journal, Vol. 7, Issue 2, June 2004.

[Sheng03] Q. Sheng, B. Benatallah, L. Zheng, M. Dumas, J. Kalagnanam, Quality Driven Web Services Composition, The 12th International World Wide Web Conference Proceedings, Eds. Yih-Farn Robin Chen, Laslo Kovacs, Steve Lawrence, ACM, New York, USA, 2003, pp. 411 - 421

[Sheth99] A. Sheth, W. M. P. Van Der Aalst, and I. B. Arpinar. Processes Driving the Networked Economy: Process Portals, Process Vortexes, and Dynamically Trading Processes. IEEE Concurrency Journal, pp. 18-31, July-September 1999.

[Sheth03] A. Sheth. Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration, invited talk at

	WWW 2003 Workshop on E-Services and the Semantic Web, Budapest, Hungary, May 20, 2003.
[Sirin03]	E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, April 2003.
[Sivashanmugam03a]	K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards, Intl. Conf. on Web Services, Las Vegas NV, June 2003.
[Sivashanmugam03b]	K. Sivashanmugam. The METEOR-S Framework for Semantic Web Process Composition. Master Thesis, Computer Science, University of Georgia, 2003.
[Sora01]	I. Sora, and F. Matthijs. Automatic Composition of Software Systems from Components with Anonymous Dependencies, Technical Report CW 314, Leuven, Belgium, May 2001.
[Srivastava03]	B. Srivastava, and J. Koehler. Web Service Composition – current solutions and open problems. ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy, June, 2003.
[Tosic01]	V. Tosic, D. Mennie, and B. Pagurek. On Dynamic Service Composition and Its Applicability to E-business Software Systems. Workshop on OO Business Sol. ECOOP, Budapest, Hungary, 2001.
[Verma04]	K.Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management (to appear, 2004).
[Zeng03]	L. Zeng, B. Benatallah, and M. Dumas. Quality Driven Web Services Composition. WWW2003, May 20-24, 2003, Budapest, Hungary.
[Zhang03]	R. Zhang, I. B. Arpinar, and B. Aleman-Meza. Automatic Composition of Semantic Web Services. Intl. Conf. on Web Services, Las Vegas NV, June 2003.
[Zhang04]	R. Zhang, Ontology-Driven Web Services Composition, MS Thesis, Department of Computer Science, University of Georgia, April 2004.