WEB SERVICES: COMPOSITION AND INTEGRATION FOR EASY USE

Shweta Roy¹ and Ranjit Singh²

¹Department of Computer Sciences and Engineering, Ajay Kumar Garg Engineering College, Adhyatmic Nagar, Ghaziabad, Pin Code-201009, E-mail: sguddr@gmail.com
²Department of Electronics Communication Engineering, Ajay Kumar Garg Engineering College, Adhyatmic Nagar, Ghaziabad, Pin-code- 201009, E-mail: ranjit.2000@gmail.com

ABSTRACT

Web services have become essential tools for efficient functioning of today’s corporate sector. Web services are designed to achieve seamless operation that enables sharing of information amongst isolated computer systems. Interoperability and composition of web services is an issue. There are set of standardized rules and specifications to makes them developer-friendly service systems.

Keywords: Web Services, Information Sharing, Simple Object Access Protocol, Web Services Definition Language.

1. INTRODUCTION

Information exchange and efficient communication is imperative for survival in today’s cut-throat competition prevalent on the business front. Information Technology has grown by leaps and bounds, and sustained, not because it seems savvy, but because businesses can function more efficiently via its deployment. The need for information exchange brings in another need to make this information selectively visible, and its visibility to be changed according to use.

Revolution in computerizing services of corporates gave rise to isolated computer systems in the earlier years. Each company had software developed and customized to its specific needs.

However, mergers, acquisitions, and business growths saw the need to share information stored in these isolated computer systems. The Internet did solve this problem to some extent. However, the Internet also opened many loopholes in security, making the owners of this information uneasy about the scope of their information’s availability.

Hence, it became imperative that, for better B2B (Business-to-Business) communication, these systems must have the ability to link up to each other, grant permissions through a system other than the Internet, and which would make all the systems network with each other like an Intranet.

Technical solution of this emerging need is offered by what is termed, ‘Web Services’. Web Services are structured on the already existing and well-known HTTP protocol that uses XML as the base language. This makes it a very developer-friendly service system.

However, most of the technologies such as RMI, COM, and CORBA involve a whole learning curve. New technologies and languages have to be learnt to implement these services.

Importantly, Web Services are based on a set of standardized rules and specifications which makes them more portable. Such is not the case with the technologies mentioned earlier.

2. COMPONENT TECHNOLOGIES

Consider a scenario in which one needs to locate a particular pharmacy store in nearby area. One would not go out on the road and ask every person one meets the way to the store. One might, instead, refer the Web site of the pharmacy on the Internet.

If one knew the pharmacy’s Web site, one would look it up directly and find the location through the store locator link.

If not, one would go to a search engine and type out the name of the pharmacy in the language that the search engine was meant to recognize. After getting the location, one would find the directions to the store, and then go to the store.

The structure of Web Services is also very similar. Web Services provide solutions for each of these previously described activities.

If we carefully look at the preceding example, we see that there is a requestor or a consumer — that is you. There
is also a service, the pharmacy store. The central database of information is the Internet, through which one finds location of the pharmacy. In the example, when you fire a search in the search engine, your request is wrapped in a structure, whose language is predetermined and localized, and then passed onto the server running the search engine.

In Web Services, techniques termed SOAP, UDDI, and WSDL represent the roles mentioned in these steps.

SOAP (Simple Object Access Protocol) is the method by which one sends messages across different modules. This is similar to how we communicate with the search engine that contains an index with the Web sites registered in the index associated with the keywords.

UDDI (Universal Description, Discovery, and Integration) is the global look-up base for locating the services. In the example mentioned earlier, this is analogous to the index service for the search engine, in which all the Web sites register themselves associated with their keywords. It maintains a record of all the pharmacy store locations throughout the country.

WSDL (Web Services Definition Language) is the method through which different services are described in the UDDI. These map to the actual search engine in our example considered earlier.

Imagine if one had to maintain a copy of such information for each and every service available in one’s vicinity! With every change in residence or requirement, one would need to compile a new list. The preceding example is a very simplified form of the Web Services environment. But, the scope of Web Services is not limited to just a look-up service. So, how does Web Services become important for the business world?

3. BUSINESS ASPECTS
Web Services in the business world, in the most simplistic fashion, provides a mechanism of communication between two remote systems, connected through the network of the Web Services [4]. For example, in case of a merger or an acquisition, companies don’t have to invest large sums of money developing software to bring the systems of the different companies together.

By extending the business applications as Web Services, the information systems of different companies are linked. These business systems then can be accessed by using simple SOAP messages over the normal HTTP Web protocol.

For example, a manufacturing company requires some raw materials to be supplied whenever the material in stock reaches below threshold levels. These levels can be constantly monitored by the business system of a trusted supplier, and promptly replenished, without having to wait for a supervisor to notice it and generate a work order.

There are many more important uses of Web Services. These, again, depend on the requirement of a company.

4. KEY CHALLENGES AND PROPOSED SOLUTIONS
One key challenge for Web services is interoperability. Seamless interoperability is the ultimate goal that Web services aim to achieve. Interoperation moves Web services beyond the elementary framework built upon the three key standards (i.e., SOAP/WSDL/UDDI). It also helps achieve robust service compositions [9, 10].

Web services require building a foundation that would provide a sound design framework for efficiently developing, deploying, publishing, discovering, composing, monitoring, and optimizing access to Web services. The proposed Web service foundation will enable the deployment of Web Service Management Systems (WSMSs) [1, 3] that would be to Web services what DBMS have been to data.

The DBMS technology has had tremendous progress with the advent of the relational model which was instrumental in giving databases a solid theoretical foundation. Web services still need to have such a strong foundational underpinning.

The WSMS is a comprehensive framework for the Web service life cycle, including developing, deploying, publishing, discovering, composing, monitoring, and optimizing access to Web services. In such a framework, Web services are treated as first-class objects that are manipulated as if they were pieces of data.

5. WSMS ARCHITECTURE
The design of the WSMS architecture leverages the research result in DBMSs [2]. Web services will be treated and manipulated as first-class object in the proposed WSMS. The key components in this architecture are modeled after those in DBMSs. The functionality of each component aims to address the issues raised by the key dimensions. The interoperation framework consists of six subcomponents, namely communication, WS messaging, WS discovery, service registry, WS representation, and WS processes.

6. WS INTEROPERATION FRAMEWORK
The interoperation framework is at the core of the WSMS architecture. It addresses the interoperability issue of Web services through the collaboration of its six subcomponents. WS-messaging combines with an underlying communication protocol (e.g., HTTP and SMTP) to enable basic interaction with Web services. A Web service takes the incoming message as the input to one of its methods and responds with the output of the method as a returning message.

WS-representation defines the Web service interface containing a set of supported methods. It specifies the signature of each method, which is similar to IDL
The security service process descriptions ontologies) for The WS management Operation input/output data signatures e.g., Service description meta-data (business taxonomies) QoWS processes to occur. also to interact with the sources of the information, number of services. These services offer the possibility currently being complemented by an ever-increasing The wealth of information available on the internet is WSMS architecture. The services are designed to achieve seamless interoperability, the interoperation framework stays at the core of the WSMS architecture. Another key challenge is web service composition. The wealth of information available on the internet is currently being complemented by an ever-increasing number of services. These services offer the possibility not only to gain more specific types of information but also to interact with the sources of the information, changing the state of these systems and causing real world processes to occur.

Here aim is to plan for the automated discovery, composition and execution of such services in order to achieve user specified goals. The ability to perform automated service composition would revolutionize a number of application areas including e-commerce and systems integration [5, 6].

For example in online travel, a planning system capable of discovering and interacting with flight and accommodation booking services could automatically arrange business trips based on user preferences. We are building a framework for automated service composition based on the information available in service directories and interface descriptions.

In this framework we do not rely on the existence of semantic mark-up in service descriptions (as advocated by the Semantic Web), but take a client side approach to the semantic interpretation of such descriptions. At present the framework consists of a semantic type matching algorithm and a planning and execution algorithm. We envisage extending the system to handle service matching and to learn to recognize the semantics of documents from examples.

Our approach to planning for web services is a pragmatic one based on the information that is currently available in service interface definitions [7, 8]. We do not require that service providers describe their interfaces using semantic markup, nor that they limit themselves to the use of “a standard data-model”. Instead we attempt to perform planning based solely on the information that is already available in the service descriptions:

1. Operation input/output data signatures
2. Service process descriptions
3. Service description meta-data (business taxonomies)

The set of operations provided by a service can be found in its WSDL description. The input/output signature of each operation tells us what type of document needs be provided in order to execute it, as well as the types of documents that will be returned upon successful and unsuccessful execution. The signature also gives us information on possible compositions of services.

For example, if a particular service has an operation “buy Book”, which takes as input an “isdn Code”, and another operation “get ISDN” (from a different service) outputs values of the same type, then the planner may try to execute the latter in order to generate input for the former. The input/output characteristics of operations in a service give only a static description of the functionality of the service.

One can describe the fact that “select Item” and “purchase Item” are both operations provided by the service, but not the restriction that the former should precede the latter during execution. Service process definitions (described using emerging choreography standards such as WSCI and BPEL4WS) can provide such procedural information to the planner.
Service classification information is found in UDDI service registries, in which services are classified according to industry-segment, provider, location, etc. If standard classification indices are used to describe the services, then the meta-data becomes useful to the planner when searching for relevant services and for inferring similarity between them.

According to our alternative approach, we do not have any description of the real-world effects of service operations. Thus we need some way of expressing goals in the system that is independent of these effects. We can do so by describing goals as information requirements— as a type of document that the planning system needs to create.

By placing restrictions on the values of fields within the requested document, one can express a goal such as: create a “purchase Order Confirmation” document within which the value of the “purchase Item” field is the name of the desired book. In order to satisfy the goal, the system needs to provide a document (an output from a service), which is of similar type to the goal and adheres to the given constraints.

These constraints could be equalities such as “book Name” equals “Harry Potter and the Philosopher’s Stone” or numerical inequalities such as that the “price” field has value “Euro”.

A planning problem is then a combination of a goal with some “local information”, which is available to the planner to use as input when executing services. An example of such local information could be the name, address, and credit card details of the person requesting the goal. The local data together with the goal value restrictions can be seen in some way as defining the initial state of the planner.

7. TYPE MATCHING

In order to compose services from their operation definitions, we need to be able in some way to match different data types, such as the goal with various service outputs. The ability to discover matches between identical types is not sufficient for our purposes as we cannot guarantee (indeed it almost certainly not the case) that the required services will input and output types from a common schema.

Thus we need to tackle the problem of data heterogeneity, which is to decide if under some mapping the data described by one data type can be substituted for that described by another. I.e. if we can take the output produced by one service, map it, and use it as input for another service.

A data type has a set of values that can be considered as representing different possible states of the world. For example when receiving a message of type “Person” with the field “name” equal to “Peter” and “age” equal to “21”, we can interpret the message as saying that there exists a person called Peter who is 21 years of age.

And if another message arrives, this time of type “University Student” with name and age as before, we can interpret it as saying that there exists a person Peter who’s 21 and goes to university.

The second message describes a smaller set of possible worlds (interpretations) than the first, i.e. the cases where Peter is not a school student nor a worker, but a university student. Now if we needed to fulfill a goal (or provide an input to a service) of type “Person”, then an instance of the message “University Student” can be used to provide the required information.

If however we require an input of type “University Student” and have a message of type “Person” the reverse is not possible as we do not know whether or not the instance to which the message refers is a university student or a school student, and so on.

Thus in order to be able to map from one data type to another we require that the latter describes a superset of the possible worlds that can be described by the former. i.e. that the target type is a more generic version of the source (under a given mapping). Now in our algorithm, when we compare the goal type to a particular service output type, we require that which is to say that all documents conforming to the output type also conform (form a subset of those conforming) to the goal type after a certain mapping _ has been applied to them. For example, a goal such as:

```xml
<Weather>
  <Temperature type="decimal"/>
  <Location type="string"/>
</Weather>
```

with a restriction that value of field “Location” should be “Adelaide”, should match against a schema such as:

```xml
<Daily Weather>
  <Local Conditions>
    <Ambient Temperature type="decimal"/>
    <Rainfall type="decimal"/>
  </Local Conditions>
  <Address>
    <City type="City Names"/>
    <State type="State Names"/>
  </Address>
</Daily Weather>
```

where:

```xml
<simple Type name="City Names">
  <restriction base="string">
    <enumeration value="Adelaide"/>
    <enumeration value="Melbourne"/>
    <enumeration value="Perth"/>
  </restriction>
</simple Type>
```
8. SERVICE COMPOSITION ALGORITHM

Having discussed an algorithm capable of matching and mapping data between heterogeneous type structures, we outline an algorithm that exploits this capability to compose and execute service operations to retrieve desired information.

Whenever we execute an operation within a service we cannot guarantee that it will execute properly, providing the desired output (e.g. where “location” equals “Adelaide”). So we are forced to interleave search and execution in order to overcome this problem of incomplete knowledge regarding the domain.

The algorithm takes as input the goal to be achieved and searches a UDDI directory for all services which are capable of outputting documents of sufficient similarity to the goal, using the type matching algorithm described previously.

The service interface with the most similar output is selected first. If there is more than one implementation of that interface, the algorithm will select one of them based on meta-data values. It then attempts to execute the particular service operation that produces the desired output.

Before doing so, it must create the required input document. It starts by using the immediately available information, such as that given in the goal, the local information, and past input and output documents if they exist. If the available information is not sufficient, the algorithm must again search the outputs of other services, i.e. the procedure calls itself recursively.

Generally, not all of the data required to fill the input document will be contained in a single source, thus the process repeats on sub-elements of the input document until a complete document is produced or a search limit is exceeded.

Having generated an input document, the algorithm attempts to invoke the operation. If it does not produce the desired output, the algorithm rolls back certain decisions made when creating the input and tries again. The heuristic guiding this search can be based on the confidence the algorithm had in its decision at each point, i.e. the quality of the match. If after a “reasonable number” of attempts, the operation still can’t be executed, then the problem may be the data given as input to the previous (successfully completed) operation.

Thus the system either tries to re-execute the previous operation with different inputs, or gives up on the service altogether and searches for a new way of achieving the goal.

The search tree created by the above algorithm can be seen as an AND-OR tree, where the “OR” branches represent different ways of creating an input, and the “AND” branches represent combinations of service outputs that together produce an input.

Leaves in the tree represent data found to be available locally. The execution algorithm described above performs a bounded best-first search through the tree, where the bound sets a limit on the number of failed execution attempts allowed for completing a given sub-tree. The execution bound is decremented for each level of decent in the tree.

This algorithm assumes that all of the operations within each service are atomic, and that the service to which they belong is stateless i.e. there are no ordering constraints on the executions of operations within a given service. In some cases this assumption may be false, and the exact ordering of operations may be critical for the correct execution of services.

For example a service might require that a “login” operation is performed prior to executing a “getStockQuote” operation. The algorithm described above would never try to invoke the former operation and thus would never be able to successfully execute the latter. In some cases such information may be available however in the form of service process descriptions.

Such process descriptions may even provide additional information regarding the flow of data between operations within a service (i.e. fields in input and output documents that refer to the same value). Carman and Serafini [10], describe a more complicated algorithm that
takes service process descriptions into account when executing services.

9. CONCLUSION

Web services have become inseparable part of B2B transactions. Web Service Integration, composition and efficient interoperability is one of the key challenges. In this paper, the solutions proposed for this problem have been outlined. One of them is treating Web Services as data wherein WSMS (Web Service Management System) can be used. Another way to compose Web Service is using service and type matching algorithm. Further work is in progress on how neural networks [11] can be used for efficient choice and composition of web services. This comprises the theme of the first co-author’s MTech Thesis.

REFERENCES


