Accurate Web Service Composition Using Global Data Schema
Shoujian Yu¹  Chao Yang¹  Jiajin Le²
College of Information Science and Technology,
Donghua University, Shanghai, China, 200051
1 {Jackyysj, Davidyoung}@mail.dhu.edu.cn
2 Lejiajin@ dhu.edu.cn

Abstract. The emergence of Web Service has led to more research into Web Services composition. It has received much interest to support business-to-business or enterprise application integration and it will reduce development time and effort for new applications. But very little work has been done on the accurate and automatic Web Services composition. We first analyze the limitations of WSDL and DAML-S and point out why they don’t fit to the accurate composition. Elicited from DFD, we propose a new Web Service composition model based on the global data schema and have designed the structure of the global schema. We discuss the composition algorithm in this new model. The data type matching algorithm is given, which is the core of accurate composition. Our implementation gives a whole idea of the composition model.

Keywords. Web Service Composition, Global Data Schema, Data Type Matching, WSDL

1 Introduction

The Web Service paradigm has emerged as an important mechanism for interoperation amongst separately developed distributed applications in the dynamic e-business environment [1, 2]. Web Services are emerging as the leading technology for encapsulating business processes and sharing them with business partners. They provide the foundation for reducing the integration cost between businesses and for creating new business [3]. Web Services can be combined to realize more complex processes or composite services, which is called Web Services Composition. Seamless composition of Web services has enormous potential in streamlining business to-business or enterprise application integration, which can offer enterprises an unprecedented opportunity to increase the level of automation in their interactions with their partners [4].

Service composition has the potential to reduce development time and effort for new applications. This has led to more research into Web Services composition with many concepts and methodologies being proposed by different research groups. However, very little work has been done on the accurate and automatic Web Services composition. This paper has the following contributions. First we analyze the limitations of current Web Service technology related to composition. We must dissolve the data heterogeneity for accurate composition. Second we proposed a new composition model using the global data schema elicited from DFD (Data Flow Diagram) in Software engineering field. Data schema can be incorporated into WSDL document, so the new model does not require wider deployment of standards such as RDF and DAML. Third we give the composition algorithm for this model and the data type matching algorithm which is the core of accurate composition. By the prototype we have implemented, this new composition model is comprehensive and has good performance.

The rest of this paper is organized as follows. Section 2 briefly reviews previous research works related to our work. Section 3 discusses limitations for the current Web Service techniques. Elicited from DFD, we propose the composition model using global data schema in section 4. We discuss the composition algorithm and how to incorporate global schema into WSDL document. In section 5 we discuss the data type matching which is the key to accurate composition. A prototype demonstrates the
composition model is implemented in section 6 and section 7 concludes this paper.

2 Related work

The Web Services composition can be built statically or dynamically. Business partners may need to form permanent (long term) or temporary (short term) relationships. In the former type of relationship, components are known in advance and alliances are statically defined. The latter form of relationship does not assume an a priori trading relationship among partners. An e-service would in this case need to dynamically discover partners to team up with to execute the required transactions. Thus, this type of dynamic integration requires support for automatic partner discover and fast e-service integration.

[5] gives the first thorough survey of patterns for composition. On the one side, the business world has developed a number of XML-based composition languages. BPEL (Business Process Execution Language) [6] combines XLANG [7] by Microsoft and WSFL (Web Service Flow Language) [8] by IBM. It includes the exact control and data flow that determines when an operation can execute. It also provides programming-language like constructs (sequence, switch, while, pick) as well as graph-based links that represent additional ordering constraints on the constructs. But the language is fairly complex and it can only be used for static composition, i.e. the process interaction must be predefined before its execution. This is the same with WSCL (Web Services Conversation Language) [9], another XML-based specification language layered on top of WSDL, for use in defining conversations between service providers and consumers developed by Hewlett-Packard. The composition of the flow is still manually obtained. On the other hand, for reasoning, the Semantic Web community uses semantic annotations to define preconditions and effects of services in the Resource Description Format (RDF) [10] using terms from pre-agreed ontologies, e.g. in DAML-S [11]. In literature [12], [13], [14], [15], semantic composition are represented. But commonly, there aren’t universal ontologies that all partners will prefer to refer to, or partners will refer to different ontologies. This kind of composition is still in its infancy and unpractical in real world, which will be discussed in section 3.2 with detail.

eFlow [16] is a platform for the specification, enactment, and management of composite e-services. A composite service is modeled by a graph, which defines the execution order of the nodes in the process. The eFlow model includes the generic service node, which supports dynamic process definitions for composite services. But the requestor must be familiar with the configuration parameters in it to promise the dynamic characteristics. The traditional AI planning technique has been used for reasoning in the automatic composition, e.g. in literature [17], [18], [19], [20]. SWORD Project developed by Stanford University models services composition as a planning problem [21]. In this approach, each service is modeled as an action. Conjunction of all the condition inputs is modeled as the precondition br the action and outputs for postcondition. State transitions are defined based on preconditions of actions and a transitions leads to new states. In [18], an AI planning system (SHOP2) is used with DAML-S Web Service descriptions to automatically compose web services. But in fact, the condition input and condition output are difficult to be determined comprehensively while the service is designed [22]. Either, multi matches may be resulted from this approach and accurate composition can't be got. Composition by planning is at most a semi-automatic process. In [23], SCET (Service Composition and Execution Tool) allows for composing services statically using its designer and storing them as Web Service Flow Language (WSFL) based specifications. The problems of dynamic composition still remain.

Our approach adopts some ideas from SWORD system. We also model services with theirs inputs and outputs parameters. But the composition in our approach is based on the global data schema
elicited from Data Flow in DFD (Data Flow Diagram)[24]. The main difference between our work and existing works is that we aim at performing automatic and accurate services composition. There are basically no published works, to the best of our knowledge, on accurate composition.

3 Limitations of Current Technology Related to Composition

Web services are pieces of code available for invoking over a network. What distinguish them from traditional remote procedure calls is the support for inter-operability, discovery and description. They are platform-independent, since all messages between service requester and service provider are transferred in XML format using SOAP[25] protocol over (for example) HTTP. Web services can be discovered via UDDI (Universal Description, Discovery and Integration)[26], which specifies a registry for dynamically locating and advertising Web services. Specification of service is done using WSDL (Web Services Description Language)[27] which defines service's abstract interface. Because WSDL does not describe the semantics of each operation, the semantic web initiative at W3C has developed DAML-S for semantically Web Service description[11]. But neither WSDL nor DAML-S can provide accurate web services matching though DAML-S approach supports matching requirements at a higher level than WSDL. In this section, we will identify the limitations of services matching by Web Service standards as well as DAML-S.

3.1 Limitations of WSDL

The Web Service standards described above are technical conventions that allow parties to easily exchange information in a standardized manner. These standards solve many problems on the technical level but they do not address the semantics of web services as a whole. The following fragment of a web service description illustrates this fact:

```xml
<message name="getTemperatureRequest">
  <part name="zipcode" type="xsd:string"/>
</message>

<message name="getTemperatureResponse">
  <part name="temperature" type="xsd:float"/>
</message>

<portType name="TemperatureServicePortType">
  <operation name="getTemperature">
    <input message="tns:getTemperatureRequest" name="getTemperatureRequest"/>
    <output message="tns:getTemperatureResponse" name="getTemperatureResponse"/>
  </operation>
</portType>
```

An agent capable of WSDL can process the above data structure and is able to interpret that the service offers messages called *getTemperatureRequest* and *getTemperatureResponse*. It can also interpret that these messages are used by a request/response operation called *getTemperature*. Furthermore it can determine the name and XML Schema types of the parameters *zipcode* and *temperature*. But it will not be able to figure out the actual meaning of these operations, e.g. that the service described will take Zone Improvement Plan (ZIP)-code of a North American city and will return
the temperature of the air in that city expressed in units of Fahrenheit.

Adding semantic information to syntactical Web Service definitions can help to better interpret the purpose and usage of Web Services, leading to a higher level of flexibility. Thus a DAML+OIL based framework providing such well-defined ontologies for semantic markup of Web Services, DAML-S occurs, but it also has many problems. The following section gives a thorough discussion of DAML-S.

3.2 Limitations of DAML-S

DAML-S aims at providing a set of ontological constructs which enable to automatically discover, evaluate and invoke web services. A couple of top-level ontologies are used to describe various aspects of service needed to express its aim and usage. They are expressed by a "ServiceProfile", a "ServiceModel" and a "ServiceGrounding". A ServiceProfile contains information needed to get a high level overview of the purpose of a service, its general input and output, its requirements (preconditions like membership or financial liquidity) and the effects of its execution. It can provide non-functional aspects of the service like guaranteed levels of quality and it can also provide other information about the service provider. A ServiceModel provides a more detailed description of the operations provided by the web service. It allows the description of a web service in terms of a set of processes and the input, output, preconditions and effects of each of the processes. ServiceModels may also contain statements about the runtime behavior and interaction patterns of processes by defining workflow constructs like conditional switches, loops or parallel executions.

This kind of modeling enables both the fine-grained description of a web service on the lower level of operations and message-parameters as well as behavior as a whole, but there are also many limitations as follows:

First, the inputs/outputs are accompanied with semantic information, but this information is described by a resource pointing to an element of ontology and the current state of ontologies is in total lawlessness. Perhaps there are many different ontologies describing the exact same thing unfortunately.

Second, the comparison of preconditions and effects fields is not applicable. Both of those fields are totally dependant on the rule representation. However, the current release of DAML-S does not support rule representation. Both preconditions and effects could represent anything, vigorously refusing to follow any formalism. Consequently, the matching engine is not able to comprehend and compare these values.

Third, it is obvious that it is really difficult for a matching engine to make safe conclusions about the compatibility of services based on these non-functional attributes.

The DAML-S is still in its infancy and a lot of work has to be done in order to overcome its limitations and problems, e.g. the lack of formalism, the intense dependency on the ontology type and content and the non-deterministic behavior, etc. It is extremely important for every reference to web resources or objects to use the same ontology. If every service description used its own private ontology, probably it would lead to a more precise description of the service, but the aggregation and composition would be practically impossible, since every member would use an “incompatible” private language. One obvious solution is to limit the allowed ontologies, but this would generate a series of critical cascading problems, questioning the value of the whole idea of the semantic web. Thus so many endeavors to automatic Web Services composition are not so successful as imaginary.
3.3 Domain Description Heterogeneity

The problem of matching for web services is further complicated by the fact that the information describing the domain is distributed (infinite), heterogeneous and incomplete. Heterogeneity is a problem both in terms of the data that services input and output, and the descriptions of the services themselves. The problem of data heterogeneity is due to the fact that there is no common high-level data model between services. In some cases service providers will adopt a common set of industry specific data types. In most cases however, an amount of heterogeneity in data descriptions will remain because the schemas defining these documents are written by different people from different systems.

One way to avoid this problem is to assume that there are standards which define the particular domain within which the planner will be working (such as standard documents for defining e-commerce in a particular vertical industry). In most domains, however, there will be no global standard for describing the world. We use global schema to resolve data heterogeneity in this paper. The rest of this paper gives a detailed discussion.

4 Accurate Web Services Composition Model

Web Service has emerged as an important mechanism for developing distributed applications in the dynamic e-business environment. Much functionality can be contained in one web service, and each is implemented by an operation. Compared with web service, data flow diagram (DFD) [24] has many similarities with Web Service. In this section, we will first discuss the similarities between Web Service Model and DFD. From this and the analysis of limitations for current techniques in section 3, we built the accurate composition model. Then we will discuss the composition algorithm and how to incorporate global schema into WSDL document.

4.1 Similarities between Web Service and DFD

DFD were introduced and popularized in the late 1970s. As one of the primary tools for structured analysis and design, DFD shows the flow of data from external entities into the system, shows how the data move from one process to another, as well as its logical storage. It is still considered one of the best modeling techniques for eliciting and representing the processing requirements of a system.

Fig. 1 is an example of DFD. Customer is an external entity. It is a source or destination of a data flow which is outside the area of study. Only those entities that originate or receive data are represented on a business process diagram. Counter is a process. It shows a transformation or manipulation of data flows within the system. Account/information is a data flow. It shows the flow of information from its source to its destination. Information always flows to or from a process. Account details is data store which is a holding place for information within the system.

The result of DFD is a series of diagrams that represent the business activities in a way that is clear and easy to communicate. Initially a general diagram is a simple representation of the entire system under investigation. This is followed by a detailed level diagram which provides a detailed analysis of
the general business process description. A general process is commonly divided into several small processes, i.e. several simple processes composed complicate process, just as Fig. 2 signifies.

Fig. 2. Business Process Composition Example in DFD

From the above analysis, we summarize several similarities and dissimilarities between Web Service and DFD.

DFD illustrates how data is processed by a system in terms of inputs and outputs. This is the same with operations of web service, of which inputs and outputs are described in WSDL.

A process generally includes several functionalities as the data flow is transacted from inputs to outputs. Also a web service can involve several operations.

Process in high level DFD is decomposed into multi sub-processes. If each sub-process is implemented by a service, the general process can be implemented by the composition of services. DFD is a tool in software engineering. It is an excellent communication tool for analysts to model processes and functional requirements. Web Service is architecture for distributed application. A web service is a software component in the application system. They are different concepts for use. But if we can built some mapping tools from DFD to Web Service, e.g. process in DFD to a service and data flow in DFD to i/o of its operations, development will be greatly streamlined from design to deployment.

4.2 Web Service Composition Model

As discussed in section 3, there must be some means to dissolve the heterogeneous problems of current Web Service standards for composition. Elicited from DFD, we use the data flow in DFD model as global data schema to provide a homogeneous background for Web Services Interface matching. A web service can be expressed as a set of operations. An operation is specified by its name, its input and output message types, i.e. o: =<name, t_in, t_out>. Web services composition is in fact the matching of inputs/outputs of operations in web services. Under a uniform schema, we can get accurate matching, i.e. accurate web service composition.

4.2.1 Composition Architecture

WSDL (Web Service Description Language) is the current standard of the description of Web Services. The syntax of WSDL is defined in XML Schema. A WSDL document describes the location of a web service, its available operations and their associated messages and data types as well as the format of their result values. The XML Schema types of message parameters may be defined using a <types> element. A <message> element is needed to compose such data types into messages. Messages need to be grouped into operations, which may define an <input>, an <output> and a <fault> message. We force that all parameters in inputs/outputs part must refer to the global schema. The structure is shown in Fig. 3.
By the global schema, we can guarantee that one service or several services will input and output types from a common schema. We can't organize all data types in one business into a global schema, because it will become unreadable. The general process is divided into several detailed processes in DFD, e.g. in Fig. 2, thus these web services implementing these processed can reference to a local schema and all the local schemas assemble as the global schema.

4.2.2 Structure of Global Schema

The global schema is the grounding for services matching. Before publishing a service, the developer should examine the global schema, and find if there has been an appropriate data type for him. If there doesn't exist the element he wants, he must add an element into the global schema before use it. The element structure in the global schema can be represented as the following schema, i.e. Fig. 4. RelatedOperation is the operation referencing to it and ioType determines whether this element is in the input parameters or in the output parameters. SemanticRelationship represents the elements that semantically related to it, which will be discussed at length in section 5. Description gives a random comment to this element for better understanding of it. Publisher can write any description relevant to this element. Requestor can query element through it.
The next section will discuss the accurate data type matching algorithm.

### 4.2.3 Incorporating global schema into WSDL

WSDL documents can incorporate information from other WSDL documents using an `<import>` element. This allows modularization of WSDL descriptions and stimulates the reuse of common abstract messages, operations and data types. Incorporating global schema into WSDL must not violate the structural integrity of a WSDL file, because SOAP containers, development environments and common Web Service applications dealing with WSDL documents would be unable to process such documents. Therefore we use an annotation method as the following code example. A graphical representation of this semantic annotation is shown in Fig. 5. The annotated WSDL description may be referenced by a t-Model entry of a UDDI registry, which provides necessary backward compatibility to existing solutions.

```xml
<message name="getTemperatureRequest">
  <part name="zipcode" type="xsd:string" GlobalMapping="SchemaName; ElementName"/>
</message>

<message name="getTemperatureResponse">
  <part name="temperature" type="xsd:float"/>
</message>
```

![Fig. 5. Incorporating global schema into WSDL document](image)

### 4.3 Web Services Composition Algorithm

A service is composed of several operations. The input/output 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. It also gives us information on possible compositions of services. For example, if a particular service has an operation “buyBook”, which takes as input an “isdnCode”, and another operation “getISDN” (from a different service) outputs values of the same type, then we know that the latter operation is a composition of the former operation. This process can be implemented automatically. Sometimes a goal needs to be composed recursively. The detailed algorithm is as follows.

In order to express the service requestor’ need, we use a set of output to express the goal, i.e.
S(O₁,O₂,...,Oₙ). The service requestor may provide some local information. For example, someone may want to query the temperature of Beijing in China, the location, i.e. Beijing, is an input for the objective web service. Thus we express this kind of local information as a set of input, i.e. S(I₁,I₂,...,Iₘ). The algorithm executes as follows.

The algorithm takes S(O₁,O₂,...,Oₙ) as the goal to be achieved and searches operations whose outputs are of sufficient similarity to the goal, using the type matching algorithm described in section 5. The operation with the most similar output is selected first. If there are more than one operations (e.g. the same weather information service is provided by two different companies), the algorithm will select one of them with the least number of inputs in S(I₁,I₂,...,Iₘ).

If the total outputs can’t be included in any operation, the operation with the maximum number of the total operations is selected and the left outputs are used as the goal for searching for operations with the same algorithm.

If S(I₁,I₂,...,Iₘ) can satisfy these services found in the above procedure, i.e. their inputs are included in S(I₁,I₂,...,Iₘ), the algorithm terminates and the goal is accomplished by the composition of these web services. Otherwise, the algorithm takes inputs or the left inputs excluded from S(I₁,I₂,...,Iₘ) of these web service as the goal to searches for operations and repeats the above procedure until S(I₁,I₂,...,Iₘ) can satisfy these found services or a search limit is exceeded, i.e. the algorithm calls itself recursively. Then the request is composed by all these web services.

5 Accurate Web Services Matching

In order to compose services based on their service descriptions we need to be able to do the data type exact matching. The significance of the accurate services composition in our method lies in the accurate data type matching from service outputs to service inputs. The ability to match data between identical types is not sufficient for our purposes. 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 matched 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.

5.1 Semantic Relationship between Data

Data type means the concept of real world. Here we take a scenario to illustrate the type matching requirements. We assume that datatype Vehicle is a more general concept of datatype Bicycle. Now if we need to fulfill a goal (or provide an input to a service) of type Vehicle, then an instance of the message Bicycle can be used to provide the required information. If however we require an input of type Bicycle and have a message of type Vehicle the reverse is not possible as we do not know whether or not the instance to which the message refers is a Bicycle or a Car, and so on. Thus in order to be able to map from one data type to another, we require some mapping, e.g. the target type is a more generic version of the source.

Before data type mapping, we need identify semantic relationships between concepts or objects, i.e. semantic relationships between data types to solve matching as the above scenario. In our approach, we identify the following three semantic relationships between terms: Synonymy, Hypernymy/Hyponymy and Meronymy [28].

- **Synonymy.** Type T₁ is a synonym of type T₂, denoted by S(T₁,T₂), if T₁ is in the synonym-set of T₂. For example, S(automobile, car).

- **Hypernymy/Hyponymy.** Type T₁ is a hypernym of type T₂, denoted by H(T₁,T₂), if T₁ is more
generic than \( T_2 \). For example, \( H(\text{car}, \text{vehicle}) \).

- **Meronymy.** Type \( T_1 \) is a meronym of type \( T_2 \), denoted by \( M(T_1, T_2) \), if \( T_1 \) is a part of \( T_2 \). For example, \( M(\text{first name}, \text{name}) \) and \( M(\text{last name}, \text{name}) \).

In order to keep the relationships between data types, we need maintain a semantic dictionary, denoted by \( D(T_1, T_2, M) \), where \( T_2 \) is the related type to \( T_1 \), and \( M \) is one of the above semantic relationships. Given a type, we use the semantic dictionary to get its synonyms, hypernyms and meronyms. The semantic dictionary is not built in advance. We can’t guarantee the semantic relationship we find is complete, which will result in inaccurate matching. The service publisher incrementally builds it when he (she) publishes a web services.

5.2 Data Type Abstraction

In order to define a type matching algorithm we first introduce a simple abstraction on the XML Schema type model. Types in XML Schema can be either primitive, simple or complex.

- **Primitive types** such as “string” and “decimal” are defined by a qualified name (a namespace URI and a local label) and the set of possible values (denoted \( \text{VAL} \)) that an instance of the datatype can take.

\[
\text{pt} := <\text{qname}, \text{VAL}>; \text{qname} := <\text{namespace}, \text{label}>
\]

- A simple type is defined in terms of a primitive type and a restriction function, which limits the range of values of the type.

\[
\text{st} := <\text{qname}, \text{pt}, \text{res}()>; \text{res} : \{\text{pt} \cdot \text{VAL}\} \rightarrow \{\top, \bot\}
\]

- Complex types are defined recursively as tree structures. Nodes in the tree structure may contain references to other primitive, simple or complex types, and can have natural language labels associated with them. (Below we denote a single node by \( n \), and a set of nodes by \( N \).)

\[
\text{ct} := <\text{qname}, N>; |N| \geq 2; n := [t<n\cdot\text{label}, N>]
\]

5.3 Data Type Matching Rules

We develop the type matching algorithm based on the idea that the target type needs to be shown to be semantically related to the source. In our type matching algorithm, when we compare the goal (target) type \( t_{\text{goal}} \) to particular service output (source) type \( t_{\text{out}} \), we require that \( t_{\text{goal}} \) and \( t_{\text{out}} \) have the above semantic relationships, i.e. \( S(t_{\text{goal}}, t_{\text{out}}), H(t_{\text{goal}}, t_{\text{out}}), M(t_{\text{goal}}, t_{\text{out}}) \).

We outline three rules used to decide whether one type (the goal document) is semantically related to another type (the output document), denoted by “\( \equiv \)” symbol. In the following, \( n \) is used to represent a node, which could be a simple or complex type, or any node within a complex type.

- **Rule 1:** let \( n_1=<l_1, \text{pt}_1> \) and \( n_2=<l_2, \text{pt}_2> \), if \( l_1 \equiv l_2 \) and \( \text{pt}_1=\text{pt}_2 \) then \( n_1 \equiv n_2 \)

  Rule 1 states that if two nodes made up of a label and a primitive type are such that the label associated with the first is semantically related to the second, and both nodes have the same primitive type, then the first node can be mapped to the second.

- **Rule 2:** let \( n_1 \in N \) and \( n_2 \in \{l_1, N_2\} \), if \( \exists n_1 \in N_2 \mid n_1 \equiv n_1 \) then \( n_1 \equiv n_2 \)

  Rule 2 states that if one of the children associated with a node is semantically related to another node, then so is the parent node. This rule allows us to “skip” intermediate nodes in the hierarchy of the more specific type.

- **Rule 3:** let \( n_1=<l_1, N_1> \) and \( n_2=<l_2, N_2> \), if \( l_1 \equiv l_2 \) and \( \forall n_1 \in N_1 \exists n_2 \in N_2 \mid n_1 \equiv n_1 \) then \( n_1 \equiv n_2 \)

  Rule 3 states that if the label associated with a node is semantically related to another node, and for all of the child nodes of the first there is also a node semantically related to a node among the
children of the second, then the first is semantically related to the second.

6 Implementation

In this section, we provide an initial prototype to illustrate the composition model we proposed. The WSDL parsing tool is in progress. We use relational database to simulate elements of the WSDL document. We design three tables as follows.

- **Element** (Name, Operation, i/oType, Description) is used for simulating global data schema. An element is associated with one or several operations and an expressive description.
- **Webservice** (ServiceName, Operation) is used for representing the operations for a service.
- **Semantics** (Name, RelatedElement, SemanticType) is used for storage of semantic relationship between elements.

![Fig. 6. Web Services composition prototype](image)

The prototype interface is demonstrated in Fig. 6. The composer should provide publisher with the global schema. In this prototype, user can query by web service (name), element or description. In the right tree view, it provides a level structure for representation. The first level is web service, and the second is the associated operations, then the related inputs/outputs parameters and their semantic related elements. User can get a whole idea from the view.

User can input composition goal parameters in the left-corner textbox. For example, we take the scenario in a travel agency as an example [29]. User inputs *Itinerary* and *Apartment* as the goal for flight reservation and hotel reservation. The prototype generates two web services as shown in the tree view. One is *FlightService*, and the other is *HotelService*. The composer also provides the ability of modifying the model. In the prototype, user can modify every node with a mouse-click on it, add or drop any node, drag a node to associate it with other node, etc.

7 Conclusions and Future Work

In this paper, we provide a new model for accurate and automatic web services composition. Accurate composition requires resolving the data heterogeneity. We summarize the limitations of some current technologies relating to accurate service composition. WSDL document only provides technical description for web services. DAML-S requires a universal ontology library, which is unpractical in real world. DFD in Software Engineering has much similarity with Web Service. We propose the new composition model using global data schema elicited from data flow in DFD. The global schema can be incorporated into WSDL document. This can provide necessary backward compatibility to existing solutions. Based on input/output of web service operation, we analyze the composition algorithm. The
key of accurate composition lies in the accurate data type matching. We put forward the matching rules by employing the semantic relationship between elements. The prototype we implement gives a whole understanding of the model.

While good results were obtained using our method, there is room for improvement. The organization of global schema needs to be improved. We will use DHT \footnote{Distribute hash table} to design a catalog frame for effective schema organization. With development of WordNet lexical reference system \footnote{WordNet lexical reference system}, we will use it for semantic inference combined with semantic dictionary. The toolkit that can incorporate schema into WSDL document and process this kind document automatically for composition is in progress.

\section*{Acknowledgements}

This work has been partially supported by The National High Technology Research and Development Program of China (863 Program) under contract 2002AA4Z3430.

\begin{thebibliography}{9}
\bibitem{1} David O'Riordan. Business Process Standards for Web Services, \url{http://www.webservicesarchitect.com}, April 2002.
\bibitem{3} Interactive Web Services Architecture White Paper. \url{http://www.webcollage.com}
\bibitem{5} B. Benatallah, M. Dumas, M.-C. Fauvet, and F. Rabhi. Towards Patterns of Web Services Composition. Patterns and Skeletons for Parallel and Distributed Computing. Springer Verlag, UK, November 2002.
\bibitem{9} Arindam Banerji, Claudio Bartolini, etc. Web Services Conversation Language (WSCL) 1.0. Hewlett-Packard Company, \url{http://www.w3.org/TR/wscl10/}, March, 2002.
\bibitem{10} RDF: Resource description framework. \url{http://www.w3.org/RDF/}
\bibitem{11} DAML-S Coalition. DAML-S: Web Service Description for the Semantic Web. ISWC01, 2002.
\bibitem{17} Srivastava, B. Automatic web services composition using planning. Proceedings of 3rd Internation


29 http://www.w3.org/2002/04/17-ws- testcase
