Automatic Service Composition Based on Enhanced Service Dependency Graph

Zhifeng Gu*, Juanzi Li*, Bin Xu*
*Department of Computer Science and Technology
Tsinghua University, Beijing, 100084, China
Email: {gzf, ljz, xubin}@keg.cs.tsinghua.edu.cn

***Abstract*— Service dependency graph (SDG) is an AND/OR graph showing input output dependencies among service operations. As dependencies in an SDG are indirectly expressed by reasoning on data models used by service interface definitions, their re-usability and expressiveness are limited. In this paper, we propose an enhanced version of service dependency graph, namely SDG+. SDG+ enhances SDG with explicit dependency declaration, which expresses dependencies directly with static explicit declarations. Based on SDG+, we developed our automatic service composition algorithm for WS-Challenge 2007, which wins the championship of composition efficiency in the competition.**

I. Introduction

Service-oriented computing (SOC), which has been widely accepted as the next generation programming paradigm, defines promising technologies that enable future computing models over the Internet. Among various service tasks, service composition plays a central role. The research of service composition covers a variety of topics, among which automatic composition is a big branch. Automatic service composition (a.k.a service synthesis) aims to create service compositions that can satisfy given constrains such as temporal behaviors [1] [2], pre/post conditions [3] [4], or input/output documents [5] [6].

The automatic composition problem defined by WS-Challenge [1] requires the inputs of a service operation to be satisfied by the outputs of other service operations, which is quite fit to be modeled by a service dependency graph (SDG) proposed by Liang [5]. After applying SDG to solve the WS-Challenge problem, we noticed that, as dependencies in SDG are implied by data model used by service interface definitions, we have to construct dependencies dynamically in our algorithm by reasoning on the data model, which occupies a big part of the overall time cost. It is obvious that according to one dataset (i.e. a set of services), the corresponding SDG is fixed, so the process of dependency construction can be shared among every run of the algorithm on this dataset. Based on this idea, we propose the concept of explicit dependency declaration, which records dependencies as static declarations and allow them to be re-used in subsequent runs of the algorithm.

At the same time, we noticed that the expressiveness of implied dependency is very limited. This is can be explained from the following aspects: 1) it may introduce many dependencies that make no sense; 2) it cannot assign customized semantics to a dependency; 3) it cannot distinct dependencies among abstract service interfaces and dependencies among service instances; 4) it does not well support dependencies that require message transformation. While all these problems may be solved by explicit dependency declaration, so we think explicit dependency declaration would not only benefit our algorithm for WS-Challenge, but also other services tasks that are using data dependencies among services.

The rest of this paper is organized as follows. In section II we give a brief introduction to SDG. Before introducing SDG+, we first define some concepts in section III. In section IV we address the problems exposed in SDG, and give the corresponding solutions in SDG+. The application of SDG+ is also discussed in this section. Section V gives an XML representation for explicit dependency declaration. Section VI introduces our automatic service composition algorithm for WS-Challenge 2007, which takes advantage of explicit dependency declarations in SDG+. Finally, in section VII, we draw a conclusion.

II. A Brief Introduction to SDG

SDG [5] [7] is a AND/OR graph that shows all the possible input-output dependencies among different services. Within SDG, services are modeled as operation nodes with an input set and an output set, which contains attributes as their elements. Attributes are abstraction of data entities/objects. An attribute can be a simple attribute or a composite attribute. Composite attribute is composed of a set of simple attributes and composite attributes, while simple attribute is logically atomic. Using SDG, for each service, we can find all the services that produce at least one attribute in its input set. Correspondingly, we can find all the services that take at least one attribute in its output set as the input attribute.

Figure 1 gives an illustration of an SDG, represented in an AND/OR graph. In the figure, the attribute node, represented with a circle, is OR node, which means all the directed edges connected to it are logically ORed. The operation node, represented with a rectangle, is AND node, which means all the directed edges connected to it are logically ANDed. An intuitive explanation is that all the input data have to be available before the operation can be performed. On the

[1] http://www.ws-challenge.org/

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.68

other hand, a particular attribute node can be produced by any operation node that has the attribute node in its output set.

Regarding composite attribute, it can be decomposed into sub-attributes which can then be fed into operation nodes. However, there is no such an assumption that the aggregation of all the sub-attributes is semantically equal to the composite attribute, so a composite attribute cannot be solved by solving all of its sub-attributes. For example, in figure 1, *a5* cannot be solved by solving *a7* and *a8*.

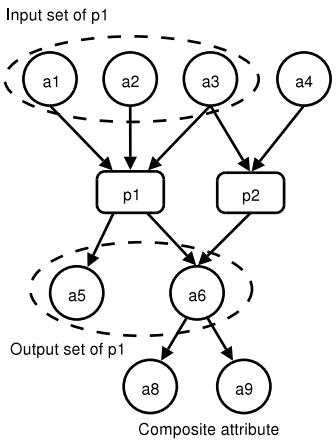


Fig. 1. An example of SDG

Given a set of known attributes and a set of required attributes, Liang has introduced a search algorithm to construct composite service templates, which is a subgraph of the AND/OR graph. The search algorithm is started from the *starting node*, which is an AND node, and is connected with all the required attribute nodes. It terminates at the *termination node*, which connects to all the known attribute nodes, and is considered to be solved.

The algorithm will find a solution graph with minimal cost (a minimal number of operation nodes and data nodes). When a solution graph is found, it is presented to the requester for evaluation. If the requester rejects the solution, the search algorithm will be applied again to find another solution graph. If no solution graph can be found or the requester rejects all the solution graphs, some "to-be-explored" operation nodes that can directly or indirectly produce required attributes will be added. After the operation nodes are added, the search algorithm will be applied again.

As a summary, in Liang's work, the procedure to find a composite service consists of two parts: 1) an automatic search algorithm and 2) human evaluation. Thus, it is called a semi-automatic procedure.

Liang's method is essentially a document-driven method for service composition, and it mainly focuses on the efficiency of the search algorithm. This is quite similar to the works [8] [9] [10] of WS-Challenge[2]. WS-Challenge defines syntactic composition and semantic composition problems, and requires competitors to solve the problems as quick as possible. We are one of the competitors, and the work introduced in [10] is the origin of this paper.

[2]http://www.ws-challenge.org/

III. Concepts of Dependency

In this section, we will discuss and define the concepts of dependency. In our work, we define two types of dependency: 1) attribute dependency and 2) operation dependency. First, we give the definition of attribute dependency.

Definition 1: An attribute dependency is a relationship between an operation and an attribute, or between two attributes. It has three forms:

- An attribute dependency from attribute a to operation p, which means p produces attribute a, denoted $p \rightarrow a$.
- An attribute dependency from operation p to attribute a, which means p consumes attribute a, denoted $a \rightarrow p$.
- An attribute dependency from a' to a through transformation T, which means attribute a' can be obtained from a by applying T on a, denoted $a \xrightarrow{T} a'$.

Corresponding to attribute dependency, operation dependency is a relationship between two operations. It actually indicates a data flow from one operation to another. We give the following definition of operation dependency.

Definition 2: An operation dependency is a data flow between two operations. It consists of five parts:

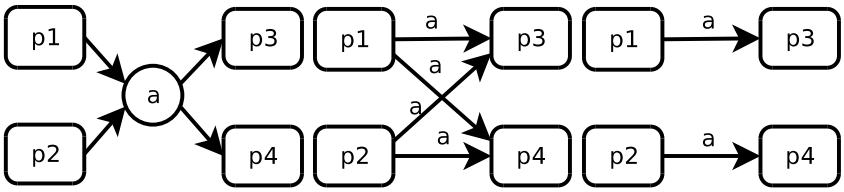
- A producer operation p'.
- A consumer operation p.
- An attribute a' produced by p'.
- An attribute a required by p.
- An attribute transformation T that transform a' to a.

We denote it as $p' \xrightarrow{a'Ta} p$. If $a' = a$, the denotation can be simplified as $p' \xrightarrow{a} p$.

In Liang's work, the concept of "dependency" is similar to the definition of operation dependency in this paper. However, Liang uses the concept of composite attribute instead of attribute transformation as introduced in section II. It is obvious that to extract a sub-attribute from a given attribute is a special form of attribute transformation. So we think that attribute transformation is a more general way than composite attribute to express the relationship between two attributes. In practice, attribute transformation can be implemented as XSLT or XPATH.

So far we have given the definition of attribute dependency and operation dependency. Note that, in service composition, what we are really interested in are operation dependencies, so, for conciseness, the term "dependency" will be used to indicate operation dependency in the following text. This is consistent to the convention in Liang's work.

From the definition, we can see that given the following data dependencies: $p' \rightarrow a'$, $a' \xrightarrow{T} a$ and $a \rightarrow p$, we can construct a dependency $p' \xrightarrow{a'Ta} p$. Furthermore, given a set of attribute dependencies, denoted D_a, we can construct a set of dependencies, denoted $P(D_a)$, by applying the above rule. Figure 2(a) and 2(b) gives an example of D_a and $P(D_a)$ respectively. We say this is an implicit way to



(a) D_a (b) $P(D_a)$ (c) D_p

Fig. 2. Attribute dependency and operation dependency

specify dependencies, and dependencies in $P(D_a)$ are called implicit dependencies. Correspondingly, an explicit way to specify dependencies is to give a set of dependencies directly, denoted D_p, and the dependencies in D_p are called explicit dependencies. It is obvious that the expressiveness of the explicit way is stronger than that of the implicit way. For example, figure 2(c) gives a dependency set D_p that cannot be expressed in an implicit way. On the reverse direction, as $P(D_a)$ itself is a set of dependencies, it can always be expressed in an explicit way.

The implicit way for dependency specification is actually based on such an assumption that if two operations are syntactically matched, the data flow between them should work. However, in practice, this assumption is very likely to be broken due to some interoperability issues. So we think explicit dependency declaration is necessary in real world applications.

IV. SDG+: The Enhanced SDG

A. Problems and Solutions

In this section we will introduce SDG+, the enhanced SDG. First, we will discuss our motivation, the problems exposed in SDG.

First SDG lacks of quantification for attributes. In SDG, there are two types of attribute: simple attribute and complex attribute. Simple attribute is a data entity/object that has a system pre-defined primitive data type, and complex attribute is composed of a set of simple and composite attributes. We think the expressiveness of this data model is not strong enough to handle certain requirements. For example, there are two operations: one provides a hotel search service that returns a list of *hotel* objects; the other provides a hotel reservation service that takes a *hotel* object as the input. In this case, the relation between a *hotel* object and a list of *hotel* objects can not be expressed. As a result, if there is not an operation that takes the list as the input and returns one *hotel* object from the list, these two operations are unable to be composed, although actually they are highly interrelated.

Second, there is no mechanism to explicitly specify dependencies. In SDG, all the dependencies are specified in the implicit way. As discussed in section III, implicit dependency specification is a very coarse-grained mechanism to express dependencies. An operation producing attribute a will introduce a dependency with every operation consuming a, so that some general attributes, such as *address* and *dimension*, will lead to a large number of dependencies, many of which are meaningless. In order to solve this problem, Liang proposed the concept of "service category" to reduce the number of implied dependencies. On the other hand, implicit dependency specification may lose dependencies that actually exist. For example, suppose $a1$ and $a2$ are two different attributes of the same concept/ontology, in SDG, an operation producing $a1$ will not be recognized as a possible precedent of the operations consuming $a2$. To avoid this problem, Liang assumes "an integrated ontology space", so that each service operation name and data entity name in SDG represents a unique concept in the integrated ontology space. We think the underlying cause of these two problems is that these is not a mechanism to explicitly specify dependencies (including attribute transformations). In our work, we allow explicitly specifying dependency in service definition. This will be introduced in detail in the following text.

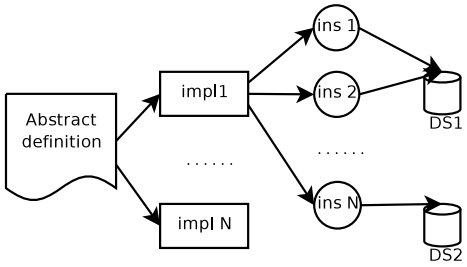


Fig. 3. Definition, implementation, instance and data-source

Third, the issues on instance-level are not concerned in SDG. Generally speaking, an abstract service definition may have several implementations, and each implementation may have several instances deployed, as shown in figure 3. Thus, it is possible that some combinations of service instances will fail to work as expected, due to:

- **The broken conformance of implementations.** As the abstract definitions of real-world services are complicated, it is usually unable to completely check the conformance of an implementation. Without sufficient testing, an arbitrary combination of service instances of different implementations cannot be guaranteed to work properly, even though they should work from the viewpoint of the abstract design.
- **Unshared data source.** The back-end data sources of different service instances may be different across com-

panies and organizations. For example, an ID generated by a service of company A is very likely to be invalid in the scope of the services of company B.

In Liang's work, these issues are not explicitly addressed, instead, an abstract service definition, its implementations and its instances are considered as a whole, but not distinguished.

Aiming the problems listed above, we propose the following enhancements respectively.

First, we define the following quantifiers for attribute:

- 1 (one and only one)
- + (one or more)
- * (zero or more)
- ? (optional, zero or one)

These quantifiers are the same as those of RELAX-NG and DTD. Thus, in SDG+, each directed edge has a quantifier associated with it, as shown in figure 4(a). If no quantifier is specified, the default quantifier is 1 (one and only one). With this enhanced data model, the relationship between *hotel* (a hotel object) and *hotel** (a list of hotel objects) can be formally expressed. As a result, SDG+ is able to exploit more dependencies among service operations. For example, in SDG+, operations producing *hotel** may be considered as dependencies of operations requiring *hotel*, while in SDG, this is irrealizable.

Second, we allow explicit dependency declaration in service definition. Each service definition may contain a list of dependencies. In SDG+, explicit specified dependency is represented by a directed edge between two operations, as shown in figure 4(b). The start of the edge is labeled with the output attribute and the quantifier. Correspondingly, the end of the edge is labeled with input attribute and the quantifier. If the input attribute and the output attribute are not matched, an attribute transformation is required, and it should be labeled on the middle of the edge. The quantifiers of the input/output attributes are not required to be matched. Note that, in figure 4(b), both $op3$ and $op1$ may contain the specification of the dependencies between them. As a result, these two versions may be inconsistent. This brings a research topic regarding the management of dependencies.

Third, we classify dependencies into two levels: abstract-level and instance-level, as shown in figure 4(c). Abstract-level dependency is defined on abstract service definitions, while instance-level dependency is defined on service instances. We prescribe that an instance-level dependency must be an operation dependency. As instance-level dependency records the interoperability between service instances, defining attribute dependency on instance-level makes no sense.

An important feature of instance-level dependency is that we allow a modifier associated with each instance-level dependency. In figure 4(c), the modifier associated with the dependency from $p2\_ins1$ to $p3\_ins1$ is F, which means this is actually a broken dependency. We use different arrows to represent different modifiers, for example, a hollow arrow indicates F. Currently three modifiers are defined: F (Fail), S (Success) and M (Mandatory). If no modifier is specified, S is the default. F means at least one failure of the dependency has been reported. S is the negative of F. It does not guarantee the correctness of the dependency, but means no error case has been reported. M means the "to" operation exclusively depends on the "from" operation.

As a summary, in SDG+, we have the following types of dependencies:

- Instance-level dependencies, e.g. $p2\_ins1 * \xrightarrow[F]{a2}? \ p3\_ins1$ in figure 4(c).
- Abstract-level dependencies, e.g. $p2 * \xrightarrow{a2}? \ p3$ in figure 4(b).
- Dependencies implied by service definitions and data models (i.e. the original SDG)

The priority of these dependencies is from high to low. In another word, when we look up dependencies, instance-level dependencies have the highest priority, and should be searched first, and then abstract-level dependency, and last the implied dependencies.

B. Discussion

As SDG+ is an extension of SDG, Liang's search algorithm may be applied on SDG+ with slight modifications. We need to compare the quantifiers of the attributes to be produced and to be consumed, and make decision according to a given policy. Policy is defined as a boolean function $M(q_{out}, q_{in})$. If the function returns true, the input and the output are considered to be matched, and the search algorithm will continue on the current search path; otherwise, it will select another path. A simple policy maybe defined as:

$$M(q_{out}, q_{in}) = \begin{cases} true & \text{if } q_{out} = q_{in} \\ false & \text{if } q_{out} \neq q_{in} \end{cases}$$

In this case, the search result will be the same as it in SDG. A reasonable policy may be defined as:

$$M(q_{out}, q_{in}) = \begin{cases} true & \text{if } (q_{out} = +) \vee \\ & (q_{out} = * \wedge q_{in} \neq +) \vee \\ & (q_{out} = 1 \wedge q_{in} \neq + \wedge q_{in} \neq *) \vee \\ & (q_{out} = ? \wedge q_{in} = ?) \\ false & \text{others} \end{cases}$$

However, we think the application of SDG+ and explicit dependency is not limited in automatic service composition algorithms. We think explicit dependency declaration provides a way for service designers, service providers and service consumers to record and share empirical knowledge. Service designers may specify abstract-level dependencies in their designs. Service providers may specify instance-level dependencies in their published service descriptions. Finally, service consumers may extract dependencies from successful compositions, and save them into their local knowledge base.

Dependencies can be considered as a distributed service recommendation and discovery mechanism. Compared with centralized service discovery mechanisms such as UDDI, service dependency is more scalable owing to its distributed natures. As services are maintained by service providers independently, the maintenance cost is distributed to every service

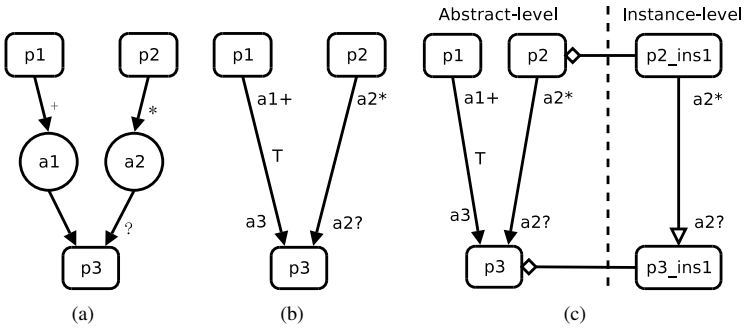


Fig. 4. The enhancements

provider. However, as the maintenance cost of the centralized service discovery mechanisms is usually centralized into one or several organizations, it will become a bottle neck when the number of services gets large.

V. XML Implementation for Explicit Dependency Declaration

A. Specify Dependency with XML

In this section, we describe how to specify dependency with XML.

As introduced in section IV-A, an explicit dependency is a dependency defined on abstract-level or instance-level. It can be generally expressed as $p'\ q'\ \xrightarrow[modifer]{a'Ta}\ q\ p$, in which, p and p' are two operations; a' is one of the outputs of p'; a is one of the inputs of p; q and q' are quantifiers of a and a' respectively; $modifer$ only makes sense when this is a instance-level dependency. These concepts are mapped to web services through the following table.

	Abstract-level	**Instance-level**
p, p'	portType/operation	service/port/operation
a, a'	part + auto-recognition	part + auto-recognition
q, q'	auto-recognition	auto-recognition
$a(a')$	XPATH or XSLT	XPATH or XSLT
$modifer$	N/A	<any>

TABLE I
CONCEPTS MAPPING

The attribute and its quantifier of a message *part* are auto-recognized by analyzing the schema. Note that WSDL supports two different binding styles: document and RPC [11]. This leads to two different styles for the definition of message *part*, shown as follows:

```
<!-- document style -->
<part name="arg" element="tns:ElementName"/>
<!-- RPC style -->
<part name="arg" type="tns:TypeName"/>
```

If definition of *TypeName* and *ElementName* is in the following form:

```
<xsd:complexType name="TypeName">
  <xsd:sequence>
    <xsd:element name="sub-item"
                 minOccurs="..." maxOccurs="..."/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="ElementName" type="TypeName"/>
```

Then, the attribute is recognized as *sub-item*, and the quantifier is recognized by applying the following rules on the *minOccurs* and *maxOccurs* attributes:

1) if (minOccurs=0 and maxOccurs=1) $q =?$
2) if (minOccurs=1 and maxOccurs=unbounded) $q = +$
3) if (minOccurs=0 maxOccurs=unbounded) $q = *$
4) otherwise $q = 1$

Otherwise, the attribute is recognized as the *part* itself, and the quantifier is recognized as 1. In practice, the structure of schema maybe very complicated. The above rules may fail to recognize quantifiers as expected. In this case, explicit declaration of attributes and the associated quantifiers will be needed.

Listing 1 gives two dependencies that are specified on instance-level and abstract-level respectively. These two dependencies are corresponding to the dependencies from $p2$ to $p3$ and from $p2\_ins1$ to $p3\_int1$ in figure 4(c). The specification of a dependency mainly consists of two parts: the *src* element and the *des* element. The *src* element specifies the source operation and attribute. Correspondingly, The *des* element specifies the destination operation attribute.

The dependencies in listing 1 are very primitive forms. In practice, we need some extended features to enhance and ease the specification of dependency. We give a brief discussion about two of them here.

The first feature is the *type* attribute of element *src* and *des*. Currently, we only support the value *webservice*, which means the enclosed description is about a web service. However, this attribute may indicate different endpoint types when being assigned other values. For example, it may indicate an XML stream processor when the value is "xmlsp". Also it may indicate a standard testing suite of a certification center to show the compatibility (or in-compatibility) against the testing suite.

The second feature is abbreviation rules for dependency specification. Defining dependency is sometimes a repetitious job, especially when there are numbers of service instances of the same service definition. In order to enable more compact specification of service dependency, we may define some

```
<dependency level="abstract">
  <src type="webservice">
    <portType>ns:portTypeName1</portType>
    <operation>Operation1</operation>
    <part>Part1</part>
  </src>
  <des type="webservice">
    <portType>ns:portTypeName2</portType>
    <operation>Operation2</operation>
    <part>Part2</part>
  </des>
  <transform>
    <xpath>/AAA/BBB</xpath>
  </transform>
</dependency>
<dependency level="instance">
  <src type="webservice">
    <service>examples</service>
    <port>pt1port</port>
    <operation>Operation1</operation>
    <part>Part1</part>
  </src>
  <des type="webservice">
    <service>examples</service>
    <port>pt2port</port>
    <operation>Operation2</operation>
    <part>Part2</part>
  </des>
  <transform>
    <xpath>/AAA/BBB</xpath>
  </transform>
  <fail/>
</dependency>
```

Listing 1. Specify dependency with XML

abbreviation rules. For example, we may omit the *operation*, *message* and *part* elements to cover all the possible dependencies between the specified services (or instances). Also we may allow each *dependency* containing multiple *src* and *des* elements, and each pair of *src* and *des* is considered as an independent dependency. This will effectively reduce the size of dependency specification.

B. Embed Dependency into WSDL

Although the specification of dependency is independent to service description, we think it is better to embed dependency into service description to reflect the relation between them. Fortunately, the extensibility of WSDL provides an easy way to do the integration.

A WSDL document can be logically divided into three parts [12]: XML schema, abstract description and concrete description. XML schema defines concepts/types in the domain, and should be fully shared and re-used. Abstract description includes definitions of *portType*, *operation* and *message*. It may import several XML schemata. Concrete description, containing definitions of *binding*, *service* and *port*, describes service instances of the imported abstract description. Obviously, the our two-level dependency model is well very matched to this logical structure. Figure 5 gives a illustration of the integration model.

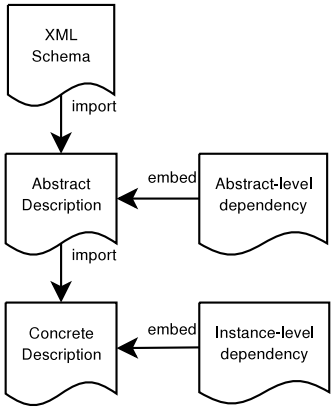


Fig. 5. Embed dependency into WSDL

Technically, the W3C schema of WSDL [13] allows certain WSDL elements containing extensibility elements. However, the WS-I Basic Profile 1.1 defines more relaxed extensibility rules. That is, every WSDL element may have extensibility elements and extensibility attributes. As we use WSDL4J, an implementation of JWSDL 1.1 that supports the WS-I extensibility rules, we can put *dependency* elements under any WSDL elements.

There are two ways to embed dependency in WSDL. The first is in-line embedding. In this way, we put dependency declarations (see listing 1) directly under WSDL elements. In principle, dependencies can be put under any element of WSDL. However, as a convention, abstract-level dependencies should be specified under *portType* or *operation*, and instance-level dependencies should be specified under *port* . The other way is specifying dependencies in external files, and import these files in WSDL. The import statement looks like,

```
<import-dep file="abstract-deps.xml"/>
```

VI. Algorithm for WS-Challenge 2007

Now, let's go back to our original motivation, to develop an high performance algorithm for WS-Challenge 2007.

A. Introduction to WS-Challenge

WS-Challenge is a competition of automatic service composition organized by annual conference of CEC/EEE. The goal of the challenge is to find all the composition solutions that satisfy given conditions as quick as possible. Each composition solution is a service chain as shown in figure 6. The following conditions are hold for every chain: 1) the input set of WS_k are contained by the union of the $input$ and the output set of WS_{k-1}; 2) the input set of WS_1 is contained by $input$; 3) the output set of WS_n contains $output$, the required document set.

B. Optimization Solution

As we introduced in section I, in order to find the precedent services of a given service in the service chain, we have to frequently lookup services according to their output documents.

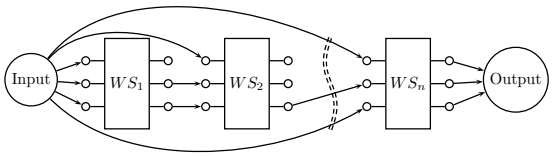


Fig. 6. Service chain

So the key to the performance of the composition algorithm lies on an operation that retrieves all the services that can produce a given document.

In this paper, we take our algorithm for WS-Challenge 2006 as the baseline algorithm. In the baseline algorithm, all the services are stored and indexed in a inverted table that uses document as its key. Given a document, the algorithm retrieves all the service that can produce it efficiently by looking up in the inverted table. For each service in the service chain as shown in figure 6, its precedents can be obtained by intersecting the looking up result of each of its input documents.

Here we use explicit dependency declaration to optimize the baseline algorithm. The process of looking up inverted table is actually a process that construct dependencies. Having the concept of explicit dependency declaration, we are able to move this process into a pre-process algorithm, and share dependencies in subsequent runs of the algorithm. In detail, we developed an pre-process algorithm that constructs input dependencies for each services, and store these dependencies into a file for reuse. For performance concern, we did not serialize dependencies as XML segments, but they are essentially the same.

With the pre-process algorithm, the optimized algorithm may eliminate the time cost for looking up the inverted table, and the performance of the algorithm will obviously be improved. Of course, the pre-process algorithm also takes time, so from the viewpoint of one time run, the optimized algorithm has no advantage; however, the output of pre-process algorithm can be reused. According to the same dataset, once the pre-process is finished, all the subsequent requests may be optimized. So, the advantage of the optimized algorithm will show with the increase of run times. According the competition rule of WS-Challenge, every request will be run 5 times, so this optimized algorithm is valid from the viewpoint of the competition.

C. Experiment Result

We use four datasets of previous WS-Challenge to evaluate the optimized algorithm. The hardware platform is Celeron 1G, 512M RAM, and the software environment is Debian Sarge. All the time is retrieved through the "gettimeofday" API.

The statistic information of the four datasets is shown in table II. The first two datasets comes from WS-Challenge 2005, the last two datasets comes from WS-Challenge 2006. WS-Challenge 2005 does not support document inheritance in XML Schema, so the "Inheritances" values of the first two datasets are zero. The pre-process time cost of each dataset is given in the last column. Note that the given value does not include the time to parse the dataset.

Dataset	Services	Inheritances	Pre-process
composition1-20-32	2156	0	25694 us
composition2-100-32	8356	0	92341 us
composition_config_small	118	1560	3918 us
composition_config_large	978	979650	298 ms

TABLE II
THE DATASETS

The experiment results are shown in figure 7. We can see that there is no significant improvement on efficiency for the first two datasets, while for the last two datasets from WS-Challenge 2006, the improvement is quite impressive. This is within our expectation. According to previous introduction, the chief difference of the optimized algorithm from the baseline algorithm is that the time cost for looking up the inverted table is eliminated, so the ratio of this part of time cost within the whole time cost determines the effect of the result.

As the datasets from WS-Challenge 2005 do not support document inheritance, the algorithm only need to look up the inverted table one time for each input document of a service in the chain, when it tries to obtain all the precedents of the service. At the same time, the inverted table is optimized with hashing, so the complexity of lookup operation is nearly a constant. As a result, the ratio of time cost for looking up the inverted table is quite small, and the advantage of the optimized algorithm vanishes.

However, the datasets from WS-Challenge 2006 support document inheritance. For each document, we need to lookup not only the services that can produce the document itself, but also all the services that can produce any documents inherited from the document. In this case, the number of lookup operation is increased rapidly. Especially in the last dataset, there are nearly one million inheritance relationships, and the inheritance tree may have thousands of extended documents. This makes the ratio of time cost for looking up the inverted table increased greatly, so this time the optimized algorithm beats the baseline algorithm with an overwhelming victory. Also we can see that, for datasets of WS-Challenge 2006, the time cost saved by the optimized algorithm is able to compensate the pre-procoss time cost (see table II) within one or two runs.

WS-Challenge 2007 supports document inheritance, and shares the datasets with WS-Challenge 2006, so this optimized algorithm is valid, and finally wins the championship of composition efficiency in the competition.

VII. Conclusion

In this paper, we propose an enhanced version of SDG, namely SDG+. We first clarify the concepts of attribute dependency and operation dependency. Then, we list the problems exposed in SDG, and propose SDG+ to solve these problems. Further, we discuss how to take advantage of SDG+ in service

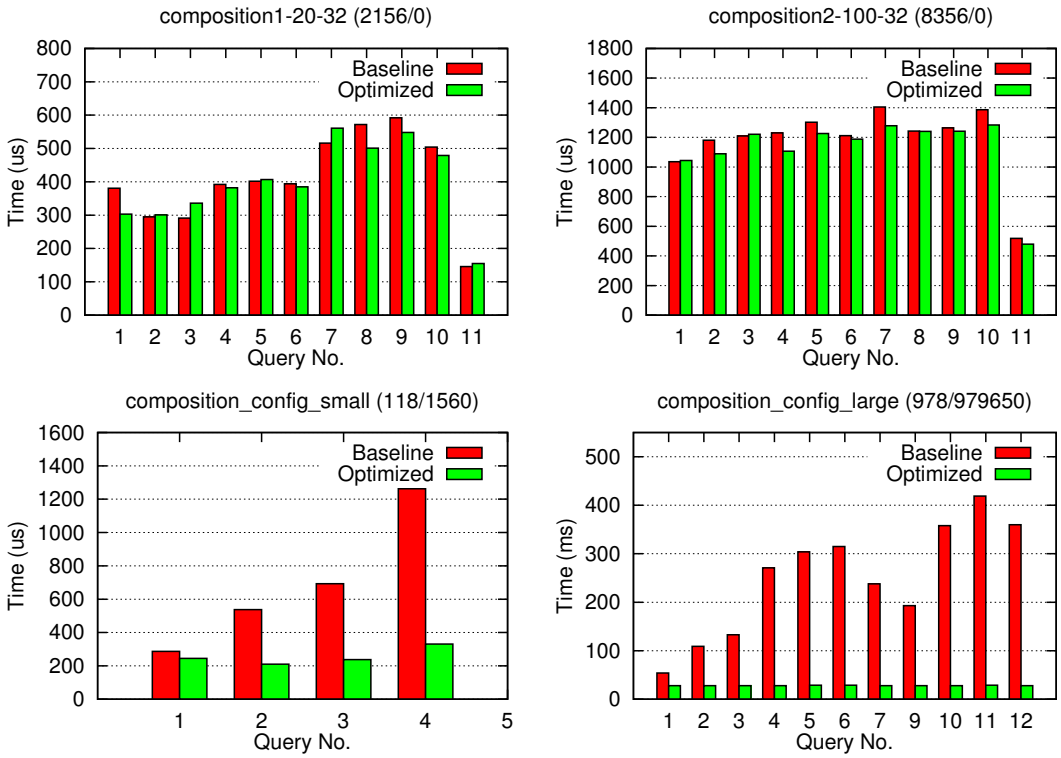


Fig. 7. Experiment Results

composition. As SDG+ is an extension of SDG, the original search algorithm based on SDG can be applied on SDG+ with slight modifications. At last, we developed an automatic service composition algorithm for WS-Challenge 2007 based on SDG+. The efficiency of this algorithm is significantly improved with the adoption of explicit dependency declaration when dealing with datasets containing complex inheritance hierarchies.

VIII. Acknowledgment

This work is supported by National Basic Research Program of China (973) under grant No.2007CB310803 and China National High-Tech Project (863) under grant No.2007AA010306. This work is also inspired by the WS-Challenge competition.

References

[1] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Automatic composition of e-services that export their behavior," in *Service-Oriented Computing - ICSOC 2003*, ser. Lecture Notes In Computer Science. Berlin: Springer-Verlag Berlin, 2003, vol. 2910, pp. 43–58.

[2] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "ESC: A tool for automatic composition of e-services based on logics of programs," in *the 5th VLDB International Workshop on Technologies for e-Services (VLDB-TES 2004)*, 2004.

[3] S. R. Ponnekanti and A. Fox, "SWORD: A developer toolkit for web service composition," in *the Eleventh International World Wide Web Conference*, Honolulu, HI, 2002.

[4] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *The VLDB Journal*, vol. 12, no. 4, pp. 333–351, 2003.

[5] Q. A. Liang and S. Y. W. Su, "AND/OR graph and search algorithm for discovering composite web services," *International Journal of Web Services Research*, vol. 2, no. 4, pp. 48 – 67, 2005.

[6] S.-C. Oh, D. Lee, and S. R. T. Kumara, "Web service planner (WSPR): An effecttive and scalable web service composition algorithm," *International Journal of Web Service Research*, vol. 4, no. 1, pp. 1–23, Jan-Mar 2007.

[7] Q. Liang, L. N. Chakarapani, S. Y. W. Su, R. N. Chikkamagalur, and H. Lam, "A semi-automatic approach to composite web services discovery, description and invocation." *International Journal of Web Services Research*, vol. 1, no. 4, pp. 64–89, 2004.

[8] M. Aiello, C. Platzer, F. Rosenberg, H. Tran, M. Vasko, and S. Dustdar, "Web service indexing for efficient retrieval and composition," in *CEC/EEE'06*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, p. 63.

[9] S.-C. Oh, H. Kil, D. Lee, and S. R. T. Kumara, "Algorithms for web services discovery and composition based on syntactic and semantic service descriptions," in *CEC/EEE'06*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, p. 66.

[10] B. Xu, T. Li, Z. Gu, and G. Wu, "SWSDS: Quick web service discovery and composition in SEWSIP," in *CEC/EEE'06*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, p. 71.

[11] R. Butek, "Which style of WSDL should I use?" 24 May 2005. [Online]. Available: http://www-128.ibm.com/developerworks/library/ws-whichwsdl/index.html?S_TACT=105AGX52\&S_CMP=cn-a-ws

[12] S. Tyagi, "Patterns and strategies for building document-based web services," Sep 2004. [Online]. Available: http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/

[13] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," March 2001. [Online]. Available: http://www.w3.org/TR/wsdl