

An Approach to Discover Dependencies between Service Operations*

Shuying Yan

Research Center for Grid and Service Computing,
Institute of Computing Technology, Chinese Academy of Sciences, 100190, Beijing, China
Email: yanshuying@software.ict.ac.cn

Jing Wang Chen Liu

Research Center for Grid and Service Computing,
Institute of Computing Technology, Chinese Academy of Sciences, 100190, Beijing, China
Email: wangjing@ict.ac.cn, liuchen@software.ict.ac.cn

Lei Liu

College of Applied Sciences,
Beijing University of Technology, 100124, Beijing, China
Email: liuliu_leilei@bjut.edu.cn

Abstract—Service composition is emerging as an important paradigm for constructing distributed applications by combining and reusing independently developed component services. One key issue of service composition is how to identify relevant service operations so as to compose services rapidly and correctly. A promising approach to simplifying the search of relevant service operations in service composition lies in the discovery of the dependencies between service operations. However, the discovery of operation dependencies is not a trivial task but a challenge. We propose an approach to discover operation dependencies in a personal problem solving environment. Our approach combines the semantic matching of inputs and outputs interfaces between service operations and the analysis of process cases to identify dependencies. The main contributions of the approach are: 1) It can be used to identify the direction of the dependencies. 2) It provides the method to measure the strength of dependencies. 3) Non-conflict property and non-redundancy property of discovered dependencies are guaranteed based on a dependency graph. Moreover, we experimentally demonstrate the efficacy of our approach by testing it under three typical bioinformatics scenarios.

Index Terms—operation dependency, service composition, process cases analysis, semantic matching

I. INTRODUCTION

Recent advances in networks, information and computation grids, and WWW have resulted in the proliferation of a multitude of physically distributed and autonomously developed Web services. Service composition is emerging as an important paradigm for

constructing distributed applications by combining and reusing independently developed component services. One key issue of service composition is how to identify relevant service operations so as to compose services rapidly and correctly.

A promising approach to simplifying the search of relevant service operations in service composition lies in the discovery of the dependencies between service operations. Since service operations cannot be considered isolated tasks. Generally, they depend on each other. It is valuable to discover the dependencies especially within a maze of interdependent service operations. However, the discovery of operation dependencies is not a trivial task but a challenge. Many researchers have significant interest in using dependencies for service composition [2]. However, most existing works assume the pre-existence of manually-constructed operation dependencies. In complex and fast evolving environments, it is practically unfeasible to obtain the dependency information and keep the information up-to-date manually. So it is necessary to detect accurate and up-to-date operation dependencies in an automatic manner.

In this paper, we propose an approach to discover operation dependencies in a personal problem solving environment. It considers not only the semantic matching of operation interfaces but also the invoking order among operations by mining of process cases. The main contributions of the approach are: 1) It can be used to identify the direction of the dependencies. 2) It provides the method to measure the strength of dependencies. 3) Non-conflict property and non-redundancy property of discovered dependencies are guaranteed in terms of a dependency graph. And corresponding algorithms are presented to preserve the dependency graph's consistency. Moreover, we have implemented the technique of dependencies discovery and have applied it in a prototypical scientific system VINCA4Science. It helps

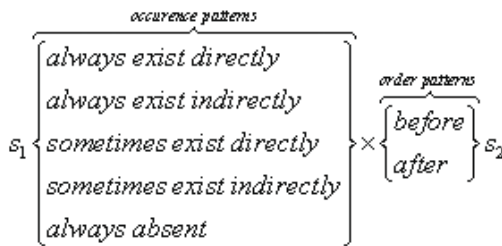
*This work is supported by Natural Science Foundation of China under Grand No. 60573117, National Basic Research Program (973) of China under Grant No. 2007CB310805 and National Hi-Tech Research and Development Program (863) of China under Grand No. 2006AA12Z202.

to ensure the correctness of composition and improve the performance.

The remainder of the paper is organized as follows: Section 2 introduces operation dependency. Section 3 presents the details of our approach for the discovery of operation dependencies. Section 4 describes and discusses the experimental result. Section 5 compares our work with other related work. Finally, we conclude in section 6.

II. OPERATION DEPENDENCY

Operation dependency relationships [7] are specified as constraints on the occurrence and order of a pair of service operations. It indicates that the occurrence of one service operation may cause the occurrence of other operation. The dependency relationships between two service operations consist of two parts: occurrence patterns and order patterns as shown in Fig. 1.



Category	Denotion
Prerequisite(R_1)	$\square s_1 \leftarrow \circ s_2$
Cause-and-Effect(R_2)	$s_2 \circ \rightarrow \square s_1$
Always-Exist-Before-Indirectly(R_3)	$\square s_1 \leftarrow s_2$
Always-Exist-After-Indirectly(R_4)	$s_2 \rightarrow \square s_1$
Sometimes-Exist-Before-Directly(R_5)	$\diamond s_1 \leftarrow \circ s_2$
Sometimes-Exist-After-Directly(R_6)	$s_2 \circ \rightarrow \diamond s_1$
Sometimes-Exist-Before-Indirectly(R_7)	$\diamond s_1 \leftarrow s_2$
Sometimes-Exist-After-Indirectly(R_8)	$s_2 \rightarrow \diamond s_1$
Always-Absent-Before(R_9)	$\square \neg s_1 \leftarrow s_2$
Always-Absent-After(R_{10})	$s_2 \rightarrow \square \neg s_1$

Figure 1. Dependency relationships between two service operations

There are four kinds of occurrence patterns:

- always exist directly: a service must occur adjacent to a given service.
- always exist indirectly: a service always occurs within a scope but not adjacent to a given service.
- sometimes exist: a service sometimes occurs within a scope.
- always absent: a service can't occur within a scope.

There are two kinds of order patterns:

- before(\leftarrow): the occurrence pattern must hold up to the occurrence of a given service.
- after(\rightarrow): the occurrence pattern must hold after the occurrence of a given service.

Then we can get ten kinds of dependency relationships between two operations as Fig. 1 shows.

III. DISCOVERY OF OPERATION DEPENDENCY

One of the most pressing needs is the ability to discover the dependencies within a maze of interdependent service operations. Discovery of operation dependencies refers to the process checking whether service operations can actually work together. The approach we take to automate the identification of dependencies mainly includes two main phrases: 1) Acquisition of operation dependency; 2) Conflict detection and redundancy check. In the following sections, we will give the details.

A. Acquisition of Operation Dependency

Acquisition of operation dependency is to find direct dependencies and indirect dependencies. It is composed of two aspects: semantic matching of operation interfaces and analysis of process cases.

We have introduced the approach of semantic matching of operation interfaces in our earlier publication [7]. In [7], semantic match degree represents the semantic match degree between the parameters of operations. It can also be measured by similarity matching between the outputs or inputs of source operation and inputs of target operation.

i) Analysis of Process Cases

Our approach for discovering the possible dependencies from process cases proceeds in two consecutive steps: 1) *Construction of the frequency table*. We use adjacency matrix to analyze direct dependency relationships, and derive indirect dependency relationships by transitive closure algorithm over adjacency matrix. 2) *Construction of the dependency table*. It is based on the metrics of frequency table. The items in the dependency table are analyzed with measures based on Pearson's Correlation Coefficient.

1) Construction of Frequency Table

The measure of reuse rate starts with the construction of frequency table. For service operation S and T the following information is abstracted out of the process cases:

- #S: the overall frequency of S directly occurs
- &S: the overall frequency of S indirectly occurs
- #T: the overall frequency of T directly occurs
- &T: the overall frequency of T indirectly occurs
- S>T: the frequency of S directly followed by T
- S<T: the frequency of S directly preceded T
- S>>>T: the frequency of S indirectly followed by T
- S<<<T: the frequency of S indirectly preceded T

An example of frequency table is given in Table 1. The frequency table can be updated incrementally when a new valuable process case is created. It is helpful for the dynamic evolution of operation dependencies. Metrics 1) through 4) is easy to achieve. Metrics 5) and 6) can be computed by adjacency matrix (directly dependency matrix). Metrics 7) and 8) can be computed by transitive

TABLE I
FREQUENCY TABLE

(T_i)	Sequence Assembly(S)							
	#S	&S	# T_i	& T_i	$S > T_i$	$S < T_i$	$S \gg \gg T_i$	$S \ll \ll T_i$
(T_1) Group Reads by Depth	140	1120	164	328	0	0	0	0
(T_2) Link Assembly Result	140	1120	256	1610	140	140	220.23	220.23
(T_3) Species Diversity Analysis	140	1120	94	1400	0	0	210.858	210.858
(T_4) Construct Scaffold	140	1120	94	1214	0	0	129.28	129.28
(T_5) Construct Superscaffold	140	1120	94	934	0	0	53.888	53.888
(T_6) Rectify Reads by Word Depth	140	1120	46	46	0	0	0	0
(T_7) Construct Chromosome	140	1120	46	328	0	0	0	0
(T_8) Query Sequencing Data	140	1120	0	0	0	0	0	0
(T_9) Word Depth Statistics	140	1120	46	0	0	0	0	0
(T_{10}) Mark Superscaffold	140	1120	94	654	0	0	0	0
(T_{11}) Rectify Reads by Manual	140	1120	94	700	0	0	7.829	7.829

closure over adjacency matrix. The details will be introduced as follows.

Construction of Adjacency Matrix

Find each explicit direct dependency between two service operations. We use adjacency matrix $D_{n \times m}$ to represent direct dependencies. In this matrix, each service operation is represented by a column and a row. If a service operation op_i is dependent on another service operation op_j , then $D[i,j] = 1$. More formally, the values of all elements in $D_{n \times m}=(d_{ij})_{n \times m}$ are defined as follows:

$$d_{ij} = \begin{cases} 1, & \text{if } op_i \rightarrow op_j \\ 0, & \text{otherwise} \end{cases}$$

AM is the zero-one matrix of this relation, where $d_{ij} = 1$ if there is a directed path from op_i to op_j and 0 otherwise. In the cases that the direct dependency relationships matrix (D) is known to be sparse, i.e. only a small subset of the elements of the corresponding array are non zero, then a sparse array implementation can be used. We can utilize the storage technique of sparse matrix. It can be stored in a compact representation format in the triple table. The above array would be represented using the form (i,j,value).

Nowadays, there are a number of web service operations that have similar or identical functionalities. In order to discover dependencies correctly, the factor should be considered. Firstly, it is necessary to group function similar service operations together. We have devoted the researches on this aspect [8]. Then we make use of the aggregation result to merge the same or similar items. Suppose that $AS=\{as_1,as_2,\dots,as_n\}$ is the set of aggregated service. And each aggregated service can be associated with a set of service operations with similar functionality $C_{asi}=\{op_{i1},op_{i2},\dots,op_{im}\}$. Let C_S be the similar items set that includes operation S and C_T be the similar items set that includes operation T. Suppose M_i^{row} be the count of non-zero elements in row i and M_j^{col} be the count of non-zero elements in column j. If $S = op_i$ and T_k

$= op_j$, we can get the following values by the analysis of D:

$$\begin{aligned} \#S &= \sum_{op_i \in C_s} M_i^{row}, & \#T_k &= \sum_{op_j \in C_T} M_j^{col}, \\ S > T_k &= \sum_{op_i \in C_s} \sum_{op_j \in C_T} d_{ij}, & S < T_k &= \sum_{op_j \in C_T} \sum_{op_i \in C_s} d_{ij} \end{aligned}$$

Example

We take the process fragments from A to F shown in Fig. 2 as an example.

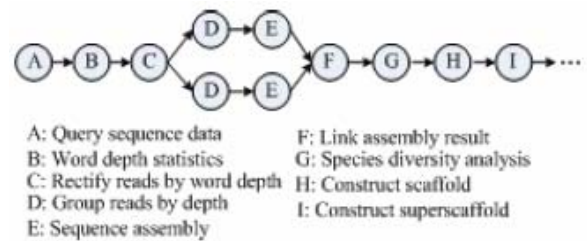


Figure 2. A Snapshot of Bombyx mori Genome Assembly Experiment

Then we can construct the adjacent matrix as:

$$D = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Utilizing the result of service aggregation, we can merge the similar items. We can add 6th column to 4th column and 6th row to 4th row. As a result, we can get the following matrix:

$$D' = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Then we can get that #RectifyReadsByWordDepth = 2, #GroupReadsbyDepth = 2, S>T = S<T = 2.

Computing of Transitive Closure Based on Adjacency Matrix

According to the description in the above section, the direct dependencies can be represented in an adjacency matrix. Because of the transitivity of dependencies, we can further to gather all indirect dependencies in the system by calculating the transitive closure of dependencies. The algorithm of calculating the transitive closure is Warshall's algorithm.

In the general case, the transitive closure t(R) of relation R can be calculated as:

$$t(R) = \prod_{i=1}^{\infty} R^i \quad R^{i+1} = R_i \circ R \quad R^1 = R$$

It can be rewritten as:

$$t(R) = \prod_{i=1}^k R^i$$

Where k depicts the length of maximum path. Then we can get the set {Rⁱ|i=2,...,k}. The termination point depends on the "depth" of the relation, i.e. on the count of vertices of the long path. Thus, the computing process may terminates before reaching i>k. It is easy to proven. Then we can get the indirect dependence relationships matrix:

$$ID = \sum_{i=2}^k (D^i \times \zeta^i), \zeta \in [0,1],$$

ζ^i depicts that the dependency degree decreases with increasing path length between S and T. Then we utilize the aggregation result to merge the same or similar items. By the analysis of indirect dependency relationships matrix (ID), we can get the following values:

$$\begin{aligned} \&S = \sum_{op_i \in C_s} \sum_{l=1}^k M_{ij}^{row}, & \&T_k = \sum_{op_j \in C_T} \sum_{l=1}^k M_{lj}^{col}, \\ S \gg \gg T_k = \sum_{op_i \in C_s} \sum_{op_j \in C_T} \sum_{l=1}^k (d_{ij} \times \zeta^l), & S \ll \ll T_k = \sum_{op_j \in C_T} \sum_{op_i \in C_s} \sum_{l=1}^k (d_{ij} \times \zeta^l) \end{aligned}$$

Example

From the adjacent matrix (D) mentioned above, we can get the indirect dependence relationships matrix by computing its transitive closures:

$$ID = \begin{pmatrix} 0 & 0 & \zeta^2 & \zeta^3 & \zeta^4 & \zeta^3 & \zeta^4 & \zeta^5 \\ 0 & 0 & 0 & \zeta^2 & \zeta^3 & \zeta^2 & \zeta^3 & \zeta^4 \\ 0 & 0 & 0 & 0 & \zeta^2 & 0 & \zeta^2 & \zeta^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \zeta^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \zeta^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Utilizing the result of service aggregation, we can merge the similar items and then get the following matrix:

$$ID' = \begin{pmatrix} 0 & 0 & \zeta^2 & 2\zeta^3 & 2\zeta^4 & \zeta^5 \\ 0 & 0 & 0 & 2\zeta^2 & 2\zeta^3 & \zeta^4 \\ 0 & 0 & 0 & 0 & 2\zeta^2 & \zeta^3 \\ 0 & 0 & 0 & 0 & 0 & 2\zeta^2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

TABLE II
CONTINGENCY TABLE

	T	-T	Row Total
S	P(T S)	P(-T S)	1
-S	P(T -S)	P(-T -S)	1
Column Total	P(T S) + P(T -S)	P(-T S) + P(-T -S)	

2) **Construction of Dependency Table**

Using the metrics in the frequency table, we can infer the direct and indirect dependencies from process cases. The result is shown in Table 2. Each cell in the dependency table can be measured based on Correlation Coefficient. Correlation Coefficient (ϕ) [5] is the computational form of Pearson's Correlation Coefficient for binary variables. The measure of Correlation Coefficient (ϕ) is

$$\phi(A, B) = \frac{P_{00}P_{11} - P_{01}P_{10}}{\sqrt{P_{0+}P_{1+}P_{+0}P_{+1}}}$$

Where $P_{i+} = \sum_{j=0}^1 P_{ij}$ and $P_{+j} = \sum_{i=0}^1 P_{ij}$.

For the other cases, we can use the analogous formula to measure. Finally, we can get the result shown in Table 3.

The aforementioned formula is often used to analyze the correlation of binary variables and the dependency direction is not taken into account. Then we specify the formula for each kind of dependencies. For example, the reuse rate of service operation S directly followed by service operation T can be measured by the following formula.

$$\phi(S_0 \rightarrow T) = \frac{P'_{00}P'_{11} - P'_{01}P'_{10}}{\sqrt{P'_{0+}P'_{1+}P'_{+0}P'_{+1}}}$$

Where

$$\begin{aligned} P'_{11} &= P(T | S) = \frac{S > T}{\#S}, \\ P'_{10} &= P(-T | S) = \frac{\#S - S > T}{\#S}, \\ P'_{01} &= P(T | -S) = \frac{\#T - S > T}{\sum_{S_i \in S_{set} \wedge S_i \neq S} \#S_i}, \\ P'_{00} &= P(-T | -S) = \frac{\sum_{S_i \in S_{set} \wedge S_i \neq S} \#S_i - (\#T - S > T)}{\sum_{S_i \in S_{set} \wedge S_i \neq S} \#S_i}. \end{aligned}$$

ii) **Combination of Two Approaches**

In order to assist end-users to discover direct and indirect operation dependencies correctly, it is necessary

TABLE III
DEPENDENCY TABLE

(T_i)	Sequence Assembly(S)			
	$S \circ \rightarrow T_i$	$S \rightarrow T_i$	$S \leftarrow \circ T_i$	$S \leftarrow T_i$
(T_1) Group Reads by Depth	-0.5402	-0.5089	-0.5349	-0.5169
(T_2) Link Assembly Result	0.9038	0.1579	0.6875	0.1965
(T_3) Species Diversity Analysis	-0.5222	0.2136	-0.5326	0.2212
(T_4) Construct Scaffold	-0.5222	0.0920	-0.5326	0.1310
(T_5) Construct Superscaffold	-0.5222	-0.1164	-0.5326	-0.0331
(T_6) Rectify Reads by Word Depth	-0.5109	-0.5014	-0.5313	-0.5170
(T_7) Construct Chromosome	-0.5114	-0.5080	-0.5313	-0.5169
(T_8) Query Sequencing Data	0	0	-0.53	-0.5170
(T_9) Word Depth Statistics	-0.5222	-0.5158	-0.5326	-0.5169
(T_{10}) Mark Superscaffold	-0.5109	0	-0.5313	-0.5170
(T_{11}) Rectify Reads by Manual	-0.5222	-0.4265	-0.5326	-0.3680

to combine the two approaches mentioned above together. Because there are some problems to discover the dependencies by using either of the approaches. On one hand, semantic matching approach can only be used to find a few of direct dependencies. And even if there exists semantic matching of operations, it does not mean there really exists a dependency. Moreover the strength of semantic match degree can't reflect the strength of operation dependency correctly. On the other hand, process cases analysis approach can find both direct and indirect dependencies, but the process cases are not complete. Some dependencies can't be inferred from process cases. Further, there may be errors in the process cases.

In order to reflect the dependency relationships between two services, we firstly introduce the notion of *composability degree*. Composability degree reflects the strength of dependency, that is, the degree to which the operations can be jointed. Here composability degree includes two fundamental factors: semantic match degree and reuse rate. The semantic match degree has been introduced in the above section. Reuse rate can be acquired from the generated dependency table. It is evaluated by mining the association rules of service operations culled from process cases.

The composability degree is defined as $w_1 \times \text{semanticMatchDegree} + w_2 \times \text{reuseRate}$. To measure the composability degree, we associate a weight to each factor. A composer can assign each weight by estimating the importance of the corresponding factor from his own point of view. If there exists the direct dependency between S and T_i , both of the weights must hold the following constraints: $w_1 > 0$, $w_2 > 0$, $w_1 + w_2 = 1$. Or else, if there exists the indirect dependency between S and T_i , the weights must hold the following constraints: $w_1 = 0$, $w_2 = 1$.

Then we can use the calculated composability degree of each kind of dependency to update the constructed dependency table. A higher composability degree means a higher likelihood there is a dependency.

Finally, we use the decision tree algorithm C4.5, which is most widely used in practice to date, to induce the

decision rules. Making use of the decision rules, we can predicate the dependencies between two operations.

B. Conflict Detection and Redundancy Check

The approach illustrated in the previous step can produce some identified dependencies between pairs of operations. And each dependency is labeled with a composability degree that expresses the strength of the inferred dependency.

However, the identified dependencies may be incorrect. The correctness of dependencies is formalized in terms of two properties: conflict-free property and non-redundancy property. The conflict-free property ensures that the satisfaction of constraints imposed by each dependency may not violate the constraints imposed by other dependencies. If a given set of dependencies has conflicts, we cannot give any assurance about the correctness of composition. The non-redundancy property ensures that no dependency specified in a set of dependencies is extraneous. The presence of extraneous dependencies in a set of dependencies unnecessarily slows down the performance.

A dependency d_x is said to conflict with another dependency d_y if the constraints imposed by d_x violate the constraints imposed by d_y . For example, consider the dependencies $ao \rightarrow b$ and $bo \rightarrow c$. If there is a new dependency $ao \rightarrow -c$, then it conflicts with the existed dependencies. Since we can infer $ao \rightarrow c$ by dependencies $ao \rightarrow b$ and $bo \rightarrow c$.

There are two kinds of dependency conflicts: 1) *Direct vs. Indirect conflicts*. For instance, consider the dependencies $ao \rightarrow b$, $bo \rightarrow c$, then dependency $ao \rightarrow c$ is conflicting with them. Because the indirect dependency can be referred from $ao \rightarrow b$ and $bo \rightarrow c$. Then there is direct versus indirect conflicts. 2) *Absent vs. Exist conflicts*. Consider the dependencies $ao \rightarrow b$ and $bo \rightarrow c$, then dependency $ao \rightarrow -c$ conflicts with them.

For example, if there is a new dependency $ao \rightarrow c$, then it is redundant for the existed dependencies. Since we can infer $ao \rightarrow c$ by dependencies $ao \rightarrow b$ and $bo \rightarrow c$.

TABLE IV
THE RESULT OF DISCOVERING DEPENDENCIES ONLY WITH SEMANTIC MATCHING

	Data Set 1				Data Set 2				Data Set 3			
	occurs	precision	recall	F-measure	occurs	precision	recall	F-measure	occurs	precision	recall	F-measure
$S \circ \rightarrow T_i$	26	0.55	0.923	0.689	57	0.456	0.95	0.616	88	0.313	0.897	0.506
$S \rightarrow T_i$	-	-	-	-	-	-	-	-	-	-	-	-
$S \leftarrow \circ T_i$	26	0.55	0.923	0.689	57	0.456	0.95	0.616	88	0.313	0.897	0.506
$S \leftarrow T_i$	-	-	-	-	-	-	-	-	-	-	-	-
unrelated	-	-	-	-	-	-	-	-	-	-	-	-

TABLE V
THE RESULT OF DISCOVERING DEPENDENCIES ONLY WITH PROCESS CASES ANALYSIS

	Data Set 1				Data Set 2				Data Set 3			
	occurs	precision	recall	F-measure	occurs	precision	recall	F-measure	occurs	precision	recall	F-measure
$S \circ \rightarrow T_i$	15	0.672	0.733	0.701	57	0.886	0.905	0.895	80	0.906	0.975	0.939
$S \rightarrow T_i$	35	0.804	0.828	0.816	152	0.980	0.967	0.973	105	0.99	0.792	0.884
$S \leftarrow \circ T_i$	15	0.672	0.733	0.701	57	0.865	0.884	0.874	81	0.863	0.929	0.894
$S \leftarrow T_i$	38	0.740	0.763	0.751	133	0.948	0.957	0.952	131	0.953	0.641	0.767
unrelated	119	0.983	0.975	0.979	890	0.990	0.994	0.992	4274	0.999	0.999	0.999

TABLE VI
THE RESULT OF DISCOVERING DEPENDENCIES WITH SEMANTIC MATCHING AND PROCESS CASES ANALYSIS

	Data Set 1				Data Set 2				Data Set 3			
	occurs	precision	recall	F-measure	occurs	precision	recall	F-measure	occurs	precision	recall	F-measure
$S \circ \rightarrow T_i$	27	0.827	0.926	0.874	84	0.932	0.965	0.948	93	0.966	0.982	0.974
$S \rightarrow T_i$	29	0.935	0.966	0.950	150	0.994	0.987	0.990	102	0.997	0.812	0.895
$S \leftarrow \circ T_i$	27	0.827	0.926	0.874	86	0.932	0.965	0.948	92	0.931	0.959	0.945
$S \leftarrow T_i$	32	0.941	0.969	0.955	115	0.963	0.963	0.963	122	0.961	0.789	0.867
unrelated	119	1	0.992	0.996	868	0.977	1	0.998	4148	0.998	0.998	0.998

Hence it is necessary to detect dependency conflicts and check dependency redundancy.

There are two kinds of dependency redundancy: 1) *Transitive redundancy*. As we all known that dependencies of same type have the transitive properties. If a dependency can be inferred by transitive deduction. 2) *Implicit redundancy*. For different types of dependencies, they also can contain implicit dependencies. For example, the dependencies $ao \rightarrow -b$ and $bo \rightarrow c$ which imply the existence of $ao \rightarrow -c$. If $ao \rightarrow -c$ exists, then there will be an implicit redundancy.

Then we propose the algorithm to automatically detect the conflicts and check the redundancies resulting from these dependencies by using the structure of dependency graph. A dependency graph is a directed acyclic graph in which nodes represent service operation and labeled directed edges represent some identified dependencies between pairs of operations. Each edge of the graph is labeled with some information (e.g. the type of dependency, semantic match degree and reuse degree) that expresses the confidence level of the inferred dependency. This will enable the users to get assurance about the correctness of the dependencies.

The algorithm includes four main steps:

Step 1: Merge the dependencies which have the same source operation and target operation. There are multiple kinds of dependencies between the same operation pair. The dependencies can be merged. As a result, we can get a composite dependency. During the process of merging,

the conflicts between dependencies should be dealt with. There are no conflicts among the dependency relationships. For two dependencies between the same operation pair, it needs to check two kinds of conflicts: Direct vs. Indirect conflicts and Absent vs. Exist conflicts. The process can guarantee the uniqueness property of dependencies between an operation pair. If there are conflicts, then notify the users. Or else, go to step 2.

Step 2: Generate initial dependency graph based on the "direct" dependency. Each "direct" dependency is depicted as an edge of the form $op_i \xrightarrow{d, \mu, \lambda} op_j$, where op_i is the source operation, op_j is the target operation, d is the type of dependency, μ is the semantic match degree, λ is the reuse rate.

Step 3: Generate the transitive closure based on the initial dependency graph. Then we can detect the conflicts of dependencies. For each type of dependency conflicts, the procedure of detecting conflict could be performed in any order, since detecting for each type of conflicts is independent. We can also check the transitive redundancy. The procedure us repeated until no more new edges can be generated.

Step 4: Generate the implicit dependency based on the initial dependency graph. The procedure is also iterative. For each pair of operations, check whether new edges could be derived by existing edges. The newly derived edges may imply more new implicit edges. They will be checked in the next round. We can use the derived result to detect the conflicts and check the implicit redundancies.

IV. EXPERIMENTAL RESULT AND EVALUATION

A. Experiment Data

In order to discover the operation dependencies by semantic matching of operation and analysis of process cases, we have collected about 100 service operations and 250 existed biological process cases. The service operations are collected from myGrid, EBI, DDBJ, NCBI, Beijing Hua Da Institute et al. The process cases are culled from several tools and organizations, e.g. taverna of myGrid, DDBJ, CNS Life Science Dept of Barcelona Supercomputing Center, Bioinformatics and Information technologies Laboratory of the University of Malaga, IBM Life Science Center of Excellent in Taiwan, Beijing Hua Da Institute et al.

B. Method

We performed preliminary experiments to evaluate the quality of our algorithm of discovering operation dependencies in terms of three key dimensions: precision, recall, and F-measure.

Precision measures the fraction of relevant items among the predicted dependencies. Precision evaluates the accuracy of the dependencies by comparing relevant items included in the predicted dependencies against the whole prediction results. *Recall* is a measure of the percentage of relevant items included in the predicted dependencies and the items that the experts have defined. *F-measure* is a trade-off between precision and recall. The metrics can be measured by the following formulas:

$$Precision = (Relevant \cap Predicted) / Predicted$$

$$Recall = (Relevant \cap Predicted) / Relevant$$

$$F - measure = (2 \times Precision \times Recall) / (Precision + Recall)$$

We test our approach under three typical bioinformatics scenarios: rice genome resequencing, bombyx mori genome assembling, and phylogenetic analysis. According to the three scenarios, we divided the operations and process cases into three categories. For each category, there are 13, 33, 67 operations respectively. There are intersections among operations. Then we classify the process cases into three groups with 40, 90, 150 process cases respectively according to the operations. There are also intersections among process cases. For each category, we compare our approach to discover operation dependencies by semantic matching and analysis of process cases with two other approaches: discovering dependencies with semantics matching and discovering dependencies with process cases analysis.

C. Results

The result of discovering dependencies by different approaches are shown in Table 4, Table 5 and Table 6. Each table shows precision, recall, and F-measure results under three scenarios respectively.

D. Discussion and Conclusion

From the results, we can see that the approach to discover dependencies only by semantic matching has a low precision. A detailed analysis of low precision shows that it dues to two main reasons. First, some operations are semantically matching but they are semantically

similar or equivalent operations and not dependent operations. Second, some operations can be jointed, however, the execution orders of operations are incorrect or conflicted. Moreover, the approach is not suitable for identifying indirect dependencies.

The approach to discover dependencies only by analysis of process cases has a higher precision than the approach by semantic matching. With the available data set increasing, the precision and recall become higher. It can be used to identify both direct and indirect dependencies. It can help to reduce the search space and alleviate users' efforts to find the correlated operations. As illustrated in Table 5, there are 67 service operations in DS₃ which will result in 4489 (67²/2) different pairs. The approach by analysis of process case detects 80(directly-after), 105(indirectly-after), 81(directly-before), 131(indirectly-before) dependencies and 4274 unrelated pairs. However, due to the limit of data set, when the operations didn't occur in the process cases, then the dependencies among the operations can't be discovered. And the errors in the process cases may bring about the incorrectness of discovery of dependencies.

Our approach combines the above two approaches. Our approach tries to avoid the deficiencies of the approaches and take the advantages of them. As Table 6 shows, the precision, recall and F-measure of our approach are higher than one of two approaches. It can be used to discover both direct and indirect dependencies.

V. RELATED WORKS

A. Representation of Dependencies

The presentation of dependencies differs in various ways due to the usage for different purposes. A common description format needs to represent the dependencies in an uniform way. Brown et al [2] and Basu et al [1] think that the dependency can be modeled as a directed edge between nodes. Each edge is labeled with a probability that expresses the confidence level of the inferred dependency. As a result, a probabilistic dependency graph will be constructed as concatenation of all identified dependencies. Ensel et al [3] argue that it is necessary to express the direction of the dependencies. Beside the direction, there are also some useful information, e.g., to express that some dependencies must occur in a certain order or to attach values of strength or likelihood.

From the above analysis, we can see that the formal representation of dependencies mainly includes some parts: 1) *order constraints*. The order constraints between two operations are not just cause-and-effect. It needs a diverse expression of order constraints. 2) *direction*. It is regarded a serious deficiency for not possible to express the direction of the dependencies. The direction may be forward or backward. 3) *strength*. The strength of a dependency indicates the likelihood that an operation is affected by another operation.

B. Acquisition of Dependencies

Ensel et al [3] have proposed an artificial neural network based technique so as to decide the existence of a dependency between objects. Unfortunately, the neural network has to be trained in a supervised manner, a laborious process and delicate process, making the practical application of the technique expensive. However, it can only detect the cause-and-effect dependency. Furthermore, they haven't touched on how to measure the strength of dependencies. To the best of our knowledge, the study of the technique's performance has not been published.

Some researchers devote to the approach based on semantic matching to explore the dependencies. Kuang et al [4] explain the importance of importing dependencies into service discovery and introduce two types of dependency between interfaces of services. However, they didn't consider the directionality, the strength of dependencies etc., while acquiring the dependencies.

The acquisition of dependencies between services is also closely related to log (workflow) mining. Steinle et al [6] have developed non-intrusive and scalable techniques to discover dependencies between components of a distributed system by mining logs. However, the approaches fail to detect the direction of dependency and distinguish the direct and indirect dependencies. The measure of dependency strength has not been mentioned, to say nothing of the properties guarantee, e.g., non-conflict property and non-redundancy property.

VI. CONCLUSION

In order to simplifying the search of relevant service operations in service compositions, we put forward an approach to discover operation dependencies in a personal problem solving environment. It combines the semantic matching of inputs and outputs interfaces between service operations and the analysis of process cases to identify dependencies. The main contributions of the approach are: 1) It can be used to identify the direction of the dependencies. 2) It provides the method to measure the strength of dependencies. 3) Non-conflict property and non-redundancy property of discovered dependencies are guaranteed based on a dependency graph.

REFERENCES

- [1] S. Basu and F. Casati. Web service dependency discovery tool for soa management. In Proc. of the IEEE International Conference on Services Computing (SCC 2007), July 2007.
- [2] A. Brown and G. Kar. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In Proc. of the IFIP/IEEE International Symposium on Integrated Network Management, page 377C390, 2001.
- [3] C. Ensel. A scalable approach to automated service dependency modeling in heterogeneous environments. In Proc. Of the Fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC '01), pages 128–139, September 2001.
- [4] L. Kuang and J. Wu. Exploring dependency between interfaces in service matchmaking. In Proc. of the IEEE International Conference on Services Computing (SCC 2007), pages 506–513, July 2007.
- [5] H. Reynolds. The analysis of cross-classifications. The Free Press, 1977.
- [6] M. Steinle and K. Aberer. Mapping moving landscapes by mining mountains of logs: novel techniques for dependency model generation. In Proc. of the 32nd international conference on Very large data bases (VLDB '06), pages 1093–1102, 2006.
- [7] S. Yan and Y. Han. Service hyperlink for exploratory service composition. In Proc. of The IEEE International Conference on e-Business Engineering (ICEBE 2007), pages 581–588, October 2007.

Shuying Yan received her MS degree in the School of Computing Science and Technology from Shandong University in 2005. She is presently a Ph.D. candidate in the Institute of Computing Technology of the Chinese Academy of Sciences. Her research interests include software integration and service grid, exploratory service composition.

Jing Wang received her Ph.D degree from the Institute of Computing Technology of the Chinese Academy of Sciences. Her research interests include software integration and service grid, personal grid workflow.

Chen Liu received his MS degree and Ph.D degree from the Institute of Computing Technology of the Chinese Academy of Sciences. His research interests include ontology learning and semantic web.

Lei Liu received his Ph.D degree from the Institute of Computing Technology of the Chinese Academy of Sciences in 2005. His research interests include knowledge acquisition and ontology learning.