International Journal of Grid and Distributed Computing
Vol. 10, No. 2 (2017), pp.37-50
http://dx.doi.org/10.14257/ijgdc.2017.10.2.04

Data Service Hyperlink and Its Application in Automatic Data Service Composition

Feng Zhang[1], Cong Liu[1,2*], Yongshan Wei[1] and Chen Liu[3]

[1] *College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao, China 266590*

[2] *Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600MB, Eindhoven, The Netherlands.*

[3] *Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, Beijing, China 100144*
[1]*zhangfengsdkd@163.com*, [*2]*liucongchina@163.com*, [3]*w3wei@sina.com.cn*, [4]*liuchen@ncut.edu.cn*

Abstract

Steered by end users, data services-based Internet data instant integration applies data services as the unified abstraction of data sources on the Internet. End users can integrate data from multiple sources through composing services, according to their instant and personalized requirements. However, it is still challenging for users to choose and compose services manually, which may lead to poor precision and relatively low efficiency. Towards this issue, we first introduce a data service hyperlink model to fully represent uni-directional correlations between input and output parameters of data services. In order to assist users to compose data services in a rapid and accurate way, this paper proposes a method to reuse correlations of two services, in which the composition plan is generated automatically according to the service hyperlink between them. In this way, the difficulty of composing services is greatly reduced. Experimental results show that the proposed method improves the efficiency and accuracy of the Internet data instant integration to a great extent.

Keywords: *data service; data correlation; automatic service composition plan generation; instant data integration*

1. Introduction

Data Mashup is an application to assist users to integrate Internet data instantly [1], and satisfies end users' instant and personalized data requirements. There are a few typical Mashup tools, such as *Yahoo*! *Pipes*, *MashMaker* [2] in the industry community, and *iMashup* [3] in academia. In recent years, some researchers use Web services to realize data Mashup. Based on data services, an instant data integration tool named *MashRoom* [4] is proposed. This work proposes end users-oriented data services (hereinafter referred to as data service or service). *MashRoom* uses the nested table as the unified model of the service's output, and provides the visual nested table interface for each service. Users integrate data through composing services. Different from traditional Web services, end users can directly understand and use data services. Furthermore, users do not need to define service composition logic beforehand, and they only need to compose services using unary and binary operations after they choose services. Through choosing and composing data services, users can integrate Internet data instantly.

Data service compositions build correlations of data sources encapsulated by services, and realize the integration of data distributed across different sources. After data sources

*Corresponding Author: Cong Liu

are encapsulated by services, correlations of underlying sources are realized by correlations among input and output parameters of different services, which are called ***data service correlations*** (correlations, for short) in this paper. During integration, the composition of two services is a basic operation, in which users have to find correlations between services, and compose them according to these correlations. Because there is semantic heterogeneity between services' input and output parameters, the composition process is cumbersome, error prone and inefficient. One major reason is that there is no carrier for service correlations in the current method, and correlations found and built cannot be reused. Consequently, users need to rediscover and rebuild correlations between services and compose services manually, leading to a poor precision and relatively low efficiency of the integration.

To counter the above problem, we propose the model suitable for data service correlations, and the method to reuse correlations to assist users composing services. **The main contributions are**: 1) we propose the *Data Service Hyperlink* (*DSHL*) to fully present correlations between two services. 2) We put forward the method to use a *DSHL* to assist users in composing services. The key point is to get the composition plan of two services according to a *DSHL* between them. In this way, the precision and efficiency of data integration steered by end users can be greatly improved.

2. Data Service Hyperlink Model

Data services can be realized through Web services based on SOAP or REST protocol. They provide read-only operations with fixed input and output parameters. To unify later expressions, for a data service named *ds*, we denote its input parameters as ***ds.Input***, output parameters as ***ds.Output***, and the nested relation schema of *ds.Output* as ***ds.Output.R***. In addition, the set of all the atomic attributes recursively included in *ds.Output.R* is denoted as ***ds.Output.R.TAttr***. For any $att \in ds.Input \cup ds.Output.R.TAttr$, we call it an "***attribute***" of ds. We denote the output data of *ds* as ***data(ds)***, and data gotten after un-nesting the nested schema of *data(ds)* as ***datainstance(ds)***. For $t \in datainstance(ds)$, t[A](A⊆ds.Output.R.TAttr) is it's values on the attribute set A. In addition, the operation of *ds* can be abstracted as a function: $\mathbf{f_{ds}:I \rightarrow R}$($f_{ds}$, for short), and I and R are the input and output of the function, respectively. $\boldsymbol{f_{ds}(d)=d'}$ shows that the input of *ds* is d, and the output of *ds* is d'.

This section focuses on how to represent services' correlations. First, correlations should record the uni-directional associations of one service to another from the perspective of services' input and output parameters. Second, a service is loosely coupled from its correlations. That is, whether correlations exist or not, a service's own functions can't be affected. Learning from the concept of Web hyperlink, we bring forward Data Service Hyperlink to represent services' correlations.

A *DSHL* can be regarded as a directed edge of two services. The service at the starting point of this edge is called the **source data service** (source service, for short), and the service at the ending point is called the **target data service** (target service, for short). A *DSHL* needs to contain two services' data correlations, and each correlation represents a mapping between atomic attributes of the source service's output and the target service's attributes. Therefore, we call a correlation a **Mapping Relation (*MR*)**. As there is syntactic and semantic heterogeneity in input and output parameters of services, a *MR* should contains extended functions [5] for transforming values of the source service's attributes to those of the target service.

Definition 1 Mapping Relation from data service ds_1 to ds_2 is a triple: MR=<atts，attt，ops>, where atts⊆ds_1.Output.R.TAttr, attt⊆ds_2.Output.R.TAttr ∪ ds_2.Input, and ops is extended functions.

In Definition 1, *ops* transforms values of *atts* to values of *attt*. It may be the composition of multiple functions. That is, *ops*=$p_1 \circ p_2 \circ \ldots \circ p_n(p_i(1 \le i \le n)$ is an extended function), and it can express the composition of multiple functions.

In a *MR*, if |MR.atts|=m, |MR.attt|=n, *MR* is called a m:n mapping relation. According to values of m and n, *MR*s can be divided into four kinds: 1:1, 1:n, m:1, and m:n. As the m:n MR is few in reality, we focus on the former three kinds.

Figure 1 shows an example of the 1:2 *MR*. Through the extended function *Split*, each data instance's value of the attribute "Brand and model" in the service "MonitorProduct" is divided into values of two attributes, which are the same semantically with atomic attributes "Brand" and "Type" in the output parameter of the service "Inventory Information".

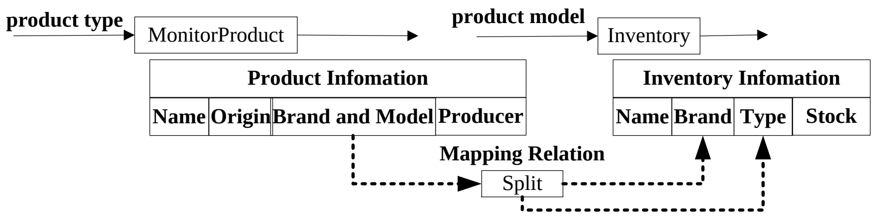

Figure 1. An Example of the 1:*n* Mapping Relation

A *MR* is divided into two types based on *MR*.attt. 1) If *MR*.attt is in the input of the target service, it is an **OI *MR***. 2) If *MR*.attt is in the output of the target service, it is an **OO *MR***. For example, Figure 1 shows an OO *MR*.

As there may be one or more correlations between the source and target service, a *DSHL* between these two services should contain one or more *MR*s to describe these correlations. Therefore, we give the definition of *DSHL* shown as Definition 2.

Definition 2 A data service hyperlink can be denoted as a tuple *DSHL* =<*linkid*, *source*, *target*, *MRS*>, where *linkid* is the identification of the hyperlink, *source* is the URI of the source service, *target* is the URI of the target service, and *MRS* ={MR_1, MR_2, ...MR_n} is *MR*s in this hyperlink.

In this paper, a *DSHL* from the source service ds_1 to the target service ds_2 is denoted as $ds_1 \rightarrow ds_2$. As mentioned, the composition of two services is a fundamental step during integration. Therefore, through assisting users to compose two services, the whole precision and efficiency of integration can be improved.

Composition of two services is closely related to the *DSHL* between them. Specifically, the way of composition can be determined by a set of *MR*s in this *DSHL*. Next, we introduce how to generate the composition plan of two services based on a set of *MR*s in a *DSHL* between them.

3. Data Service Hyperlink-based Automatic Service Composition Plan Generation

Since data service composition needs unary and binary operations, this section first introduces operations for composing services, and then presents basic lows for generating the composition plan according to *MR*s in a *DSHL*. Finally, we propose the service composition plan automatic generation method based on a *DSHL*.

3.1. Data Service Composition Operations

MashRoom defines unary and binary operations for service composition. Unary operations can filter, project, transform services' output, and so no. Binary operations can realize compositions of services. Main operations are shown in Table1.

There are mainly three kinds of binary operations for composing services ds_1 and ds_2. 1) *Connect*. When invoking ds_2, this operation can link ds_1's atomic attributes in its output and the input of ds_2, and assigns values of ds_1's atomic attributes of the output to the input parameters of ds_2. 2) *Join*. This operation can join the output of two services. When executing this operation, ds_1 and ds_2 are invoked first, and then the operation join their outputs based on their outputs' atomic attributes. 3) *Set* operations include *Union*, *Intersect* and *Difference*. The nested relation schema of data services must be the same before using these operations. Then, these operations can return the union, intersection and difference of services' outputs, respectively.

Set operations need no configuring parameters to execute, while *Connect* and *Join* need parameters according to input and output parameters of services. Each parameter is called the operation's ***configuration***, which indicates requirements for the execution of the operation. This paper extends *Connect* and *Join* presented in [5], so that they can realize more complicated composition.

Table 1. Data Service Composition Operations

Type	Operation	Function description
Unary Operations	Project	Delete attributes of the output of a service
	Filter	Filter the output of a service
	Nest	Nest the output of a service
	Un-nest	Un-nest the output of a service
	Sort	Sort the output of a service
	Rename	Rename attributes of the output of a service
	Extend	Extended function (multiple functions)
Binary Operations	Connect	Connect the outputs of one service to the inputs of another service
	Join	Join of outputs of services
	Union	Union of outputs of services
	Difference	Difference of outputs of services
	Intersect	Intersect of outputs of services

Definition 3 *Connect*'s Configurations. *Connect* composing services ds_1 and ds_2 is denoted as $Connect(<s_1,t_1,op_1>,<s_2,t_2,op_2>\ldots<s_n,t_n,op_n>)$. $<s_i,t_i,op_i>(1\leq i\leq n)$ is a configuration, where $s_i\subseteq ds_1.Output.R.TAttr$, $t_i\subseteq ds_2.Input$, and op_i is extended functions between s_i and t_i.

In Definition 3, $<s_i,t_i,op_i>$ describes requirements for execution of *Connect*: $ds_2.Input.t_i = op_i(ds_1.Output.R.s_i)$. All requirements indicated by $<s_1,t_1,op_1>$, $<s_2,t_2,op_2>\ldots<s_n,t_n,op_n>$ are $(ds_2.Input.t_1=op_1(ds_1.Output.R.s_1)) \wedge (ds_2.Input.t_2=op_2(ds_1.Output.R.s_2)) \wedge \ldots \wedge (ds_2.Input.t_n= op_n(ds_1.Output.R.s_n))$.

Definition 4 *Join*'s Configurations. *Join* composing services ds_1 and ds_2 is denoted as $Join(<s_1,t_1,op_1>,<s_2,t_2,op_2>\ldots<s_m,t_m,op_m>)$. $<s_i,t_i,op_i>(1\leq i\leq m)$ is a configuration, where $s_i\subseteq ds_1.Output.R.TAttr$, $t_i\subseteq ds_2.Output.R.TAttr$, and op_i is extended functions between s_i and t_i.

In Definition 4, $<s_i,t_i,op_i>$ corresponds to a condition for the nature join when executing *Join*: $ds_2.Output.R.t_i=op_i(ds_1.Output.R.s_i)$. $<s_1,t_1,op_1>,<s_2,t_2,op_2>\ldots<s_m,t_m,op_m>$ correspond to conditions for the nature join when executing the *Join* operation: $(ds_2.Output.R.t_1= op_1(ds_1.Output.R.s_1)) \wedge (ds_2.Output.R.t_2=op_2(ds_1.Output.R.s_2)) \wedge \ldots \wedge (ds_2.Output.R.t_m= op_m(ds_1.Output.R.s_m))$.

As we can see from above two definitions, configurations of *Connection* and *Join* are consistent with *MR*s between services. That is, *Connection* and *Join* can deal with 1:1, 1:n and n:1 correlations of two services when composing services. Next, we introduce the correspondence between *MR*s and operations, that is, how to compose services with these operations based on a set *MR*s.

3.2. Correspondence between Mapping Relations and Operations

When composing the source and target service of a *DSHL*, users can choose different combination of *MR*s in this *DSHL*. Consequently, there are different ways to compose the source and target service. For example, there are three *MR*s in the *DSHL* linking service "*getOwnInfo*" to "*getPersonInfo*" in Figure 2. Users can compose these two services as follows: 1) according to MR_1, users use *Connect*, and its configuration is "*getOwnerInfo.Output.R.IDCardNumber=getPersonInfo.Input.IdentificationNumber*". 2) According to MR_2, users first invoke these two services. Then, according to the condition "*getOwnerInfo.Output.R.IDCardNumber=getPersonInfo.Output.R.IdentificationNumber*", use *Join* operation to join outputs of two services. 3) According to MR_3, the composing way is similar with the second, and the configuration of *Join* is: "*getOwnerInfo.Output.R.Ownername=getPersonInfo.Output.R.name*". 4) According to MR_1 and MR_2. First, users compose services according to MR_1 as the first case. Second, according to MR_2, the composing result can be filtered through the unary operation *Filter*, whose filter condition is "*getOwnerInfo.Output.R.IDCardNumber= getPersonInfo.Output.R.IdentificationNumber*". From this, we can see there are different composition ways based on different combinations of *MR*s. Next, we present how to compose two services according to OI and OO *MR*s.

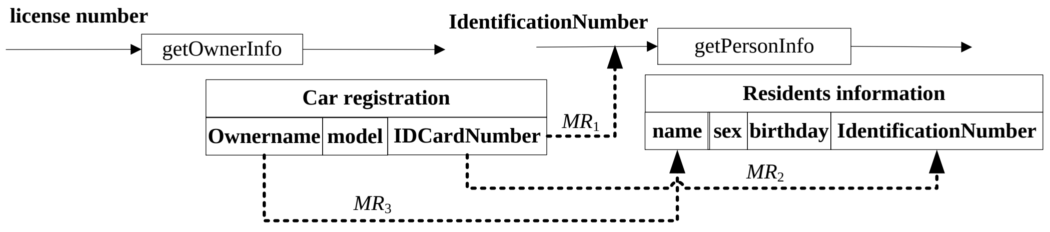

Figure 2. Examples of Mapping Relations

3.2.1. Composition Constraint and Realization of OI *MR*s:

When composing services according to an OI *MR*, users need to assign values of some attributes in the source service's output to input parameters of the target service. In this way, through invoking the target service, data associated with the source service's output can be obtained. From this, we can see that the OI *MR* represents a kind of constraint on the output of the target service. We call this constraint as ***composition constraint*** of OI *MR*s.

Definition 5 For an OI *MR* in $ds_1 \rightarrow ds_2$, its **composition constraint** is that after composing ds_1 and ds_2, $\forall d_2(d_2 \in datainstance(ds_2) \land f_{ds2}(d)=d_2) \rightarrow \exists d_1(d_1 \in datainstance(ds_1) \land d[MR.attt] = MR.ops\,(d_1[MR.atts]))$.

According to Definition 5, for an OI MR, when composing services, we demand values of attributes in *MR*.attt are the same with those of *MR*.ops(*MR*.atts). In this way, output data of the target service is constrained, and the one-way data correlation between the source and target service is realized.

Theorem 1 An OI *MR*=<*atts, attt, ops*> is in $ds_1 \rightarrow ds_2$. If *Connect*(<*MR.atts, MR.attt, MR.ops*>) is used to compose ds_1 and ds_2, this operation can realize the composition constraint of *MR*.

Proof: $\forall d_2(d_2 \in datainstance(ds_2))$, then $\exists d(f_{ds2}(d)=d_2)$, where d is the value of an input parameter in ds_2. As $d[attt]=MR.ops(d'[atts])(d' \in datainstance(ds_1))$, $\forall d_2(d_2 \in datainstance(ds_2) \land f_{ds2}(d)=d_2) \rightarrow \exists d_1(d_1 \in datainstance(ds_1) \land d[attt] = MR.ops$

(d_1[atts])). Therefore, *Connect*(<*MR.atts, MR.attt, MR.ops*>) realizes the composition constraint of *MR* when composing ds_1 and ds_2.

According to Theorem 1, the composition constraint of an OI *MR* can be realized by the *Connect* operation, and the attributes, extended functions in *MR* are parts of configurations of the operation.

However, a *DSHL* may contain multiple OI *MR*s, composition constraints of all *MR*s need to be realized when composing the source and target service with *Connect*. The method to realize composition constraints of multiple OI *MR*s is as follows.

Corollary 1 $MRS=\{MR_1, MR_2,...MR_n\}$ is a set of OI *MR*s in $ds_1 \rightarrow ds_2$, where $MR_i=<atts_i\ ,\ attt_i\ ,\ ops_i>(1 \le i \le n)$, and $MR_i.attt_i \cap MR_j.attt_j = \emptyset (1 \le i,j \le n)$. If *Connect*($<MR_1.atts_1, MR_1.attt_1, MR_1.ops_1>,\ldots<MR_n.atts_n, MR_n.attt_n, MR_n.ops_n>$) is used to compose ds_1 and ds_2, it can realize composition constraints of all *MR*s in MRS.

Proof: According to Definition 3, *Connect*($<MR_1.atts_1, MR_1.attt_1, MR_1.ops_1>,<MR_2.atts_2, MR_2.attt_2, MR_2.ops_2>\ldots <MR_n.atts_n, MR_n.attt_n, MR_n.ops_n>$) satisfies constraints: (ds_2.Input.$attt_1$=$MR_1.ops_1$(ds_1.*Output.R.atts*$_1$)) ∧ … ∧ (ds_2.Input.$attt_n$=$MR_n.ops_n$(ds_1.*Output.R.atts*$_n$)). Meanwhile, according to Corollary 1, this operation can satisfy (ds_2.Input.$attt_i$=MR_i.ops_i(ds_1.Output.R.$atts_i$)) ($1 \le i \le n$). That is, $\forall MR_i(MR_i \in MRS)$, this operation can realize the composition constraint of MR_i. Therefore, this operation can realize composition constraints of all *MR*s in *MRS*.

3.2.2. OO MRs and Operations: When users compose two services according to an OO *MR*, binary operations that can be used are *Join* and *Set* operations. For Set operations, nested relation schemas of two services must be the same. *Join* operation can execute natural join on outputs according to correlations between two services. Therefore, the follow theorem can be given when composing two services according to an OO *MR*.

Theorem 2 *MR*=<*atts, attt, ops*> is an OO *MR* in $ds_1 \rightarrow ds_2$. 1) If nested relation schemas of ds_1 and ds_2 are different, *Join*(<*MR.atts*, *MR.attt*, *MR.ops*>) can be used to compose ds_1 and ds_2. 2) If the nested relation schemas of ds_1 and ds_2 are the same, *Join*(<*MR.atts, MR.attt, MR.ops*>), *Intersect*, *Union*, and *Difference*, can be exclusively used to compose ds_1 and ds_2.

Based on the above analysis, Theorem 2 is correct, and we don't prove it in details. Next is the method to compose data services according to multiple OO *MR*s.

Corollary 2 $MRS=\{MR_1, MR_2,...MR_n\}$ is a set of OO *MR*s in $ds_1 \rightarrow ds_2$, where $MR_i=<atts_i, attt_i, ops_i>(1 \le i \le n)$. If nested relation schemas of ds_1 and ds_2 are different, *Join*($<MR_1.atts_1, MR_1.attt_1, MR_1.ops_1>,\ldots<MR_n.atts_n, MR_n.attt_n, MR_n.ops_n>$) can be used to compose ds_1 and ds_2.

The proof process is similar with that of Corollary 1, and here the details are not given.

From Corollary 1-2, we can see that for multiple OI *MR*s, one *Connect* operation can realize their composition constraints. For OO *MR*s, when nested relation schemas of the source and target service are different, one Join operation can join outputs of two services.

However, a *DSHL* may contain both OI and OO *MR*s. In this case, users need to realize composition constraints of OI *MR*s, and deal with the outputs of two services according to OO *MR*s at the same time. To this end, *Connect* needs to be executed first to make the target service output data, which satisfies constraints of OI *MR*s. After composition, their nested relation schemas are merged. At this point, binary operations cannot be used any more. For OO *MR*s, *Join* filters the composition result using the natural join condition from its configurations when merging outputs of services. If services are already composed, the composition result only needs to be filtered according to the natural join condition. Therefore, the unary operation *Filter* is the only operation needed to filter the composition result, and the filter condition is the natural join condition that *Join* operation's configurations represent. According to the analysis, the following corollary can be given.

Corollary 3 MRS={MR_{OI}, MR_{OO}} is a set of MRs in $ds_1 \rightarrow ds_2$, where MR_{OI}={MR_{OI1}, MR_{OI2},...MR_{OIn}} is OI MRs, and MR_{OO}={MR_{OO1}, MR_{OO2},...MR_{OOm}} is OO MRs. When composing ds_1 and ds_2, *Connect*(<$MR_{OI1}.atts_1$, $MR_{OI1}.attt_1$, $MR_{OI1}.ops_1$>,…<$MR_{OIn}.atts_n$, $MR_{OIn}.attt_n$, $MR_{OIn}.ops_n$>) can be used first. And then, the operation *Filter* can be used to filter the composition result, and the filter condition is ($MR_{OO1}.attt_1 = MR_{OO1}.ops_1$ ($MR_{OO1}.atts_1$)) $\wedge \ldots \wedge$($MR_{OOm}.attt_m = MR_{OOm}.ops_m$ ($MR_{OOm}.atts_m$)).

Based on above theorems and corollaries, next, we give the method to automatically generate data service composition plans according to a set of *MR*s in a *DSHL*.

3.3. Automatic Generation of Data Service Composition Plan

When composing two services linked by a *DSHL*, users can determine the composition method based on part of *MR*s in this *DSHL*. On one hand, users can edit the output of the source service, such as deleting some attributes, which causes existing *MR*s invalid. On the other hand, users can choose part of valid *MR*s by themselves as the basis for composition. Therefore, the composition method of the source and target service is not fixed. In order to assist users to compose services, this sector presents how to get the composition plan of the source and target service automatically according to a set of selected *MR*s in a *DSHL*.

Definition 6 A Service Composition Plan(SCP) of data service ds_1 and ds_2 is a set of operations ($op_1,op_2,\cdots op_n$) to be executed in order for composing ds_2 with ds_1, and op_i($1\leqslant i\leqslant n$) contains its configurations.

3.3.1. Basic Ideas and Problems: According to theorems and corollaries in 3.2, the method for generating *SCP* according to *MR*s in $ds_1 \rightarrow ds_2$ is as follows. 1) If nested relation schemas of output of ds_1 and ds_2 are the same, *Set* or *Join* operation can be used to compose services. 2) If nested relation schemas of ds_1 and ds_2's output are different, needed operations with their configurations are added into the *SCP* in order according to *MR*s. At this point, there are three cases: a) only OI *MR*s; b) only OO *MR*s; c) both OI and OO *MR*s. All these three cases can be handled to produce the *SCP* through theorems and corollaries in 3.2.

However, there still is a problem for the case that includes OI *MR*s. According to Corollary 1, for multiple OI *MR*s: MR_1, MR_2,…MR_n, *Connect* is used under the condition $MR_i.att_i \cap MR_j.att_j = \emptyset (1\leq i,j\leq n)$. The reason is that if there are duplicate input parameters of the target service in these *MR*s, when invoking the target service, assignments of the target service's input have conflicts or duplication. In this case, only part of these *MR*s can be chosen. Moreover, for the target service, it is invoked when executing *Connect*, and the more input parameters are assigned through OI *MR*s, the less input parameters are assigned manually by users. Hence, through choosing OI *MR*s, we also hope the number of the target service's input parameters assigned by the source service's output get the maximum.

3.3.2. Choosing OI MRs: This sector introduces a sub-process *ChooseOIMR*, which choose from a specified collection of *MR*s during generating the *SCP*. By *ChooseOIMR*, for the target service, the number of its input parameters included in the chosen *MR*s is the largest, and its input parameters have no duplication at the same time.

If chosen OI *MR*s are denoted as MRS={MR_1,MR_2,...MR_n}, the question can be described as follows: to get MRS'={MR_{i1},MR_{i2},...MR_{im}}(m≤n), where $MRS' \subseteq MRS$, and following conditions are satisfied:

$$\begin{cases} MR_i.attt \cap MR_j.attt = \varnothing (MR_i, MR_j \in MRS', \text{且} i_1 \le i, j \le i_m) \\ \sum_{j=1}^{m} | MR_{ij}.attt | \text{ is the maximum} \end{cases}$$

The above question focuses on *attt* in each *MR*. If *MR_i.attt* is denoted as S_i, this question can be abstracted as follows: for $M=\{S_1,S_2 \ldots S_n\}$, S_i is a collection of elements, how to get $M'=\{t_1,t_2 \ldots t_m\}(m \le n)$, $M' \subseteq M$, and following conditions are met:

$$\begin{cases} t_i \cap t_j = \varnothing(t_i, t_j \in M', \text{且} 1 \le i, j \le m) \\ \sum_{i=1}^{m} | t_i | \text{ is the maximum} \end{cases}$$

This question can be considered as how to choose mutually disjoint sets from a collection of sets, and the union of these sets contains the maximum number of elements. The approach is as follows. Since both the number of sets contained in M' and the number of elements contained in all the sets of M' are unknown, all possible set combinations can be enumerated in the case that the number of sets in M and the number of elements in each set are not large (this applies to the situation of correlations among data services). Then, the final set combination that contains the most elements can be found. Meanwhile, a strategy to reduce the size of the search space can be taken. While getting all set combinations, for a set combination containing the set t_i, all the sets intersecting with t_i are excluded from the combination any more. Therefore, the matrix *interMatrix*, where each row and each column represent a set in M, can be gotten to describe the intersection of sets in M. Specifically, if the intersection of the set i and j is not empty, *interMatrix*[i][j]=*interMatrix*[j][i]=1, otherwise *interMatrix* [i][j]=*interMatrix*[j][i]=0. With *interMatrix*, the search space is effectively reduced.

Algorithm 1 implements *ChooseOIMR*. In this algorithm, sub-process *getPlans* generates *finalPlans*, which contains all combinations of all the sets. And then, the sub-process *getMaxCoveringSets* gets chosen *MR*s from *finalPlans*.

Algorithm 1. *ChooseOIMR*

Input：a set of OI *MR*s named *MRS*
Output：a set of chosen OI *MR*s named *CMR*

```
sets = getAtttSets(MRS)  // get the set made up of attt in each MR according to MRS
interMatrix = getInterMatrix(sets)  // get interMatrix
finalPlans = getPlans(sets)
CMR = getMaxCoveringSets(finalPlans, sets)
return CMR
```

3.3.3. SCP Generation Algorithm: The algorithm *GetSCP* to generate *SCP*s of the source and target service according to a set of *MR*s in a *DSHL* is shown in Algorithm 2.

Algorithm 2. *GetSCP*

Input：a set of *MR*s named *MRS* in $ds_1 \rightarrow ds_2$, and *MROI* and *MROO* are OI and OO *MR*s set in *MRS*, respectively
Output：*SCP*s

```
if (ds1.Output.R=ds2.Output.R)
    SCP.add(Union, Intersect, Difference, Join)
if (MROI ≠ null){
    ChoosenOIMR = ChooseOIMR(MROI)
    for each mroi in ChoosenOIMR
        Connect.addParams(mroi.attt=mroi.ops(mroi.atts)) //add configurations for Connect
```

```
         SCP.*add*(Connect)  // add the Connect operation into SCP
         **if** (MROO ≠ null){
             for each mroo in OOMR
                 Filter.*addParas*(mroo.attt=mroo.ops(mroo.atts))//add configurations for Filter
             SCP.*add*(Filter)  // add the Filter operation into SCP
         }
     }**else{**
         **if** (MROO ≠ null){
         for each mroo in MROO
           Join.*addParams*(mroo.attt=mroo.ops(mroo.atts))// add configurations for Join
         SCP.*add*(Join)  // add the Join operation into SCP
         }//end if
     **}**//end else
     return **SCP**
```

4. Experimental Results and Evaluation

DSHL-based *SCP* automatic generation provides an approach to assist users composing services. It aims to improve the efficiency and accuracy of data service composition steered by end users. In this sector, we present the experiments to show the effect of this method for the efficiency and accuracy of service composition.

(1) Evaluation indicators

First, we use the **construction time** for composing two data services as the indicator for efficiency. Second, we use the **number of errors** that appear when users compose two services as the indicator for accuracy. Learning from [6], times that users consult others for the sake of errors when composing services are counted manually, and it is recorded as the value of the second indicator.

(2) Experimental data sets and the evaluation method

We find 1,023 popular APIs from Programmable Web, and take them as the data set. We use the data service encapsulation tool in our data service composition platform to encapsulate these APIs as data services, and build their service hyperlinks manually according to these services' input and output parameters. In each experiment, five pairs of services (ten services) that have *DSHL*s are randomly selected, and sorted by the ascending number (the number is bigger, and the composition is more complex) of *MR*s

To contrast the effects, users compose each services pair in the composition platforms that don't support *DSHL*s and support *DSHL*s separately. In each experiment, five users select a pair of services randomly, and each user composes the same pair of services in above two platforms. Then the construction time and the number of errors under two different platforms are measured. Above experiments are repeated 10 times, and the average of two indicators are computed. Moreover, in order to ensure that users are familiar with the services to be composed equally, users compose services in the platform not supporting *DSHL*s first. Then, users compose them in the platform supporting *DSHL*s. In this way, users are all familiar with services to be composed in two platforms.

(3) Experiment results

Experiment results are shown in Figure 3, where NS and S present the results in the platforms not supporting *DSHL*s and supporting *DSHL*s separately, and NS-S presents the difference between NS and S. As can be seen, as *MR*s in the *DSHL* increase, the construction time and number of errors increase in two cases. With the automatically generated *SCP*s, the construction time and the number of errors in N are all less than those in NS. Meanwhile, as the composition complexity increases, with the help of *SCP*s, more time can be saved for users.

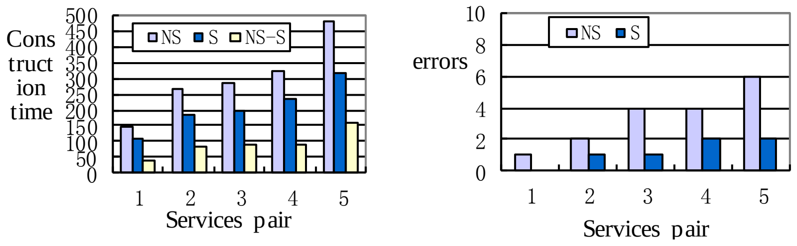

Figure 3. Experimental Results

There are two reasons for the results. Firstly, when users compose services manually, they need to set configurations for operations, and determine the composition method of services. This process is error prone. Especially when using extended functions to deal with complex grammatical and semantic heterogeneities, users have to write complex formulas, leading to more errors. However, in the *SCP*, all the formulas for the operations' configurations are generated automatically. Users only need to copy and paste configurations of the operations, and need not to write formulas manually. In addition, with the *SCP*, users can directly get the needed operations and their execution order. Users only need to choose from the plan and modify it partly, and then compose services. Secondly, according to the *SCP*, the number of input parameters asking users to input is least when invoking the target service. Therefore, the process of composing services is more efficient.

The final conclusion is that when users compose two services with a *DSHL*, with the help of the automatically generated *SCP*, construction time and errors are all reduced, and the efficiency and accuracy are all improved.

5. Related Works

This paper introduces a data service correlations model and its application in the Web data instant integration. The focus is the *DSHL* model and *SCP* automatic generation based on *DSHL*s. Related works include the correlation model of Web services' input and output parameters, and automatic Web service composition.

5.1. Correlation Model of Services Input and Output Parameters

Web services have fixed input and output parameters. If an output parameter of one service can be semantically mapped to an input parameter of another service, these two services can build correlations based on their input and output. For Web services based on SOAP, the grammar correlations between one service's output parameter and another service's input parameter are presented [7,8].

In addition, some works also learn from Web hyperlinks to build correlations of services, and present some concepts of service hyperlinks. Different service hyperlinks can be brought forward for different services, but all these service hyperlinks are built based on services' input and output parameters. For example, hyperlinks to model the behavioral constraints of services are proposed [9]. In addition, for RSS Feeds, OpenAPI, REST and SOAP services on the Internet, HyperService is put forward [10], which defines four kinds of service relations. Among these relations, the linked data relation represents data correlations of two services, which is still based on the semantic mapping between the source service's output parameters and the target service's input parameters.

The correlations of services in above works all focus on service composability, namely, the **invoking relations** between output parameters of the source service and input parameters of the target service. However, for data services used for data integration, the composition needs not only invoking service, but also aggregating services' output. Moreover, all these works only consider 1:1 mappings between services. Hence, these models can't describe correlations of data services fully.

For the data services based on the nested table, the data service hyperlink to describe correlations of data services are also put forward [11, 12]. Similar to the service hyperlink in this paper, its definition considers not only mappings between output parameters of the source service and input parameters of the target service, but also mappings between two services' outputs. However, similar to above works, these works only consider the 1:1 mapping relations between services' parameters.

Compared with above works, through *MR*s, *DSHL* in this paper can describe not only correlations between the source service's output parameters and the target service's input parameters, but also correlations of two services' output parameters. Moreover, the *DSHL* describes not only the 1:1 mapping of services' parameters, but also 1:n and n:1 mappings.

5.2. Automatic Service Composition

Automatic service composition is a main problem in service computing. Existing works include methods based on the artificial intelligence theory [13-17], such as logical inference, theorem proving, and task planning. In addition, related works also include methods based on graph search [18, 19]. For the first works, services need to be preprocessed and transformed formally. Therefore, these works are complex and hard to realize. For the second works, because the number of services in the repository is large, the construction time of services relation graph is excessively long [20].

In this paper, the automatic service composition method based on *DSHL* is used for data services. This method puts emphasis on the automatic generation of two services' composition from the view of actual operations. Therefore, from the services type, the purpose for composition, and the operability, our work is different from these works.

6. Conclusion

To improve the efficiency and accuracy of data service composition steered by end users, this paper proposes the notion of *DSHL* to present correlations of data services, and gives the application of *DSHL* in automatic data service composition. Specifically, we give the basic lows of getting the composition plan of the source and target service according to OI and OO mapping relations in a *DSHL*. Based on these basic lows, we present the automatic generation method of data service composition plan using *DSHL*. According to experimental evaluations, we argue that with the help of generated service composition plan, the efficiency and accuracy of data service composition steered by end users are greatly improved.

In the future, we will improve this work from the following aspects. This paper mainly focuses on the data services based on the nested table. As this kind of services can be realized through Web services based on REST or SOAP, next, we will extend this work and apply it to data correlations of traditional Web services.

Acknowledgments

This work is supported by Scientific Research Foundation of Shandong University of Science and Technology for Recruited Talents under Grant No. 2014RCJJ049, the Shandong Key Research Project of Statistical Research under Grant No. KT16195, the Open Fund of Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data North China University of Technology, the National Natural Science Foundation of China under Grant No. 61502281, 61602279, 61472229.

References

[1] H. Chen, B. Lu, Y. Ni, G. Xie, C. Zhou, J. Mi and Z. Wu, "Mashup by surfing a web of data APIs", Proceedings of the VLDB Endowment, **(2009)**; Lyon, France.
[2] G. D. L. Lorenzo, H. Hacid, H. Y. Paik and B. Benatallah, "Data integration in mashups", SIGMOD Record, vol. 38, no. 1, **(2009)**, pp. 59-66.
[3] X. Liu, G. Huang, Q. Zhao, H. Mei and B. M. Brian, "iMashup: a mashup-based framework for service composition", Science in China Series F: Information Sciences, vol. 57, no. 1, **(2014)**, pp. 1-20.
[4] G. Wang, S. Yang and Y Han, "Mashroom: end-user mashup programming using nested tables", Proceedings of 18th international conference on World Wide Web, **(2009)**; Madrid, Spain.
[5] Y. Wen, "User-Steered Just-in-time Integration of Multiple Heterogeneous Data-Source", Beijing: Institute of Computing Technology, Chinese Academy of Science, **(2012)**.
[6] Y. Han, G. Wang, G. J and P. Zhang, "Situational data integration with data services and nested table", Service Oriented Computing and Applications, vol. 7, no. 2, **(2012)**, pp. 129-150.
[7] Z. Gu, B. Xu and J. Li, "Service Data Correlation Modeling and Its Application in Data-Driven Service Composition", IEEE Transactions on Service computing, vol. 3, no. 4, **(2010)**, pp. 279-291.
[8] S. Chen, Z. Feng and H. Wang, "Service Relations and Its Application in Services Oriented Computing", Chinese journal of computers, vol. 33, no. 11, **(2010)**, pp. 2068-2083.
[9] S. Yan, Y. Han, J. Wang, C. Liu and G. Wang, "A User-Steering Exploratory Service Composition Approach", Proceedings of IEEE International Conference on Services Computing, **(2008)**; Honolulu, HI.
[10] C. Zhao, C. Ma, J. Zhang, J. Zhang, L Yi and X. Mao, "HyperService: Linking and Exploring Services on the Web", Proceedings of IEEE International Conference on Web Services, **(2010)**; Miami, FL.
[11] C. Liu, J. Wang, M. Zhu and Y. Han, "Discovery of Service HyperLinks with User Feedbacks for Situational Data Mashup", International Journal of Database Theory and Application, vol. 8, no. 4, **(2015)**, pp. 71-80.
[12] M. Zhu and C. Liu, "Discovery and Reuse of Service Hyperlinks for Efficient Data Mashup", Proceedings of IEEE World Congress on Services, **(2014)**; Anchorage, AK.
[13] D. Wu, B. Parsia, E. Sirin, J. Hendler and D. Nau, "Automating DAML-S Web services composition using SHOP2", Proceedings of the Second International Semantic Web Conference, **(2003)**; Sanibel Island, FL.
[14] J. Rao, P. Kungas and M. Matskin, "Logic-Based Web services composition: from service description to process model", Proceedings of IEEE International Conference on Web Services, **(2004)**; San Diego, California.
[15] R. Akkiraju, B. Srivastava, A. Ivan, R. Goodwin and T. Mahmood, "SEMAPLAN: Combining planning with semantic matching to achieve Web service composition", Proceedings of IEEE International Conference on Web Services, **(2006)**; Chicago, IL.
[16] H. Zhao and P. Doshi, "A hierarchical framework for composing nested Web processes", Proceedings of International Conference on Service-Oriented Computing, **(2006)**; Chicago, USA.
[17] J. Cheng, C. Liu, M. Zhou, Q. Zeng and A. Ylä-Jääski, "Automatic Composition of Semantic Web Services Based on Fuzzy Predicate Petri Nets", IEEE Transaction on Automation Science and Engineering, vol. 12, no. 2, **(2015)**, pp. 680-689.
[18] L. Aversano, G. Canfora and A. Ciampi, "An algorithm for Web service discovery through their composition", Proceedings of IEEE International Conference on Web Services, **(2004)**; San Diego, California.
[19] S. Hashemian and F. Mavaddat, "A graph-based framework for composition of stateless Web services", Proceedings of European Conference on Web Services, **(2006)**; Zurich, Switzerland.
[20] S. Deng, J. Wu, Y. Li and Z. Wu, "Automatic Web Service Composition Based on Backward Tree", Journal of Software, vol. 18, no. 8, **(2007)**, pp. 1896-1910.

Authors

Feng Zhang, he received the PhD degree from Shandong University of Science and Technology in 2014. He is a lecture in College of Information Science and Engineering of Shandong University of Science and Technology, Qingdao, China. His current research interests include data integration, service computing, cloud computing, and so on.

Cong Liu, he received his M.S. degree in computer software and theory from Shandong University of Science and Technology, Qingdao, China in 2015. He is currently a Ph.D candidate within the Section of Information Systems (IS) at the Department of Mathematics and Computer Science in Eindhoven University of Technology. His research interests are in the areas of Busines Process mining, Petri nets, and Software Process Mining.

Yongshan Wei, he received the PhD degree from Shandong University of Science and Technology in 2011. He is an assocaite professor in College of Information Science and Engineering of Shandong University of Science and Technology, Qingdao, China. His current research interests include machine learning, data integration, service computing, and so on.

Chen Liu, he is an associate professor at Research Center for Cloud Computing, North China University of Technology. His current research interests include data integration, service modeling, service composition, cloud computing, and so on.