Sharing large data collections using data services in cloud environment

Qi Ji, Yuanming Zhang and Gang Xiao
College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China
Hongfang Zhou
Zhejiang Xizi Fuward Electric Motor Corporation, Hangzhou, China, and
Zheng Lin
Zhejiang Academy of Equipment Science, Hangzhou, China

Abstract

Purpose – Data service (DS) is a special software service that enables data access in cloud environment and provides a unified data model for cross-originating data integration and data sharing. The purpose of the work is to automatically compose DSs and quickly generate data view to satisfy users’ various data requirements (DRs).

Design/methodology/approach – The paper proposes an automatic DS composition and view generation approach. DSs are organized into DS dependence graph (DSDG) based on their inherent dependences, and DSs can be automatically composed using the DSDG according to user’s DRs. Then, data view will be generated by interpreting the composed DS.

Findings – Experimental results with real cross-originating data sets show the proposed approaches have high efficiency and good quality for DS composition and view generation.

Originality/value – The authors propose a DS composition algorithm and a data view generation algorithm according to users’ DRs.

Keywords Data service, Data service composition, Data view, Cloud computing

Paper type Research paper

1. Introduction

With the wide and deep development of information technology over the past ten years, a large number of data sets are rapidly generated in different organizations. These data sets comprise multiple modalities with diverse representations and distributions, while requiring interactions among one another. Given the rate at which the data are produced, allowing the data to be accessed without geographical limitations will eliminate several bottlenecks in data-oriented innovations and will be especially valuable for further processing, such as big data analysis and mining.

Service computing provides a flexible computing architecture to support freely accessible abundant resources deployed on the web and has emerged as one important promising research area. Not only are various functions of software encapsulated into the services, named the web service, but diverse data produced from the software are also encapsulated into services, called data service. Data service (DS) shields heterogeneous data through a set
of access interfaces and provides a unified model for data integration and data sharing. In this case, data can be provided on the web regardless of geography. By composing DSs and presenting them as data composition views, the cross-organization data integration and data sharing can be effectively implemented.

There are already many works focused on DS encapsulation (Carey et al., 2008; Yu et al., 2017), DS access (Wang et al., 2017), DS-based mining (Zorrilla and García-Saiz, 2013), DS composition (Malki et al., 2014), data view generation (Xie and Xiao, 2014) and other aspects (Zhang et al., 2013). With the enormous explosion of DSs in recent years, an important issue is how to compose DS and generate DS view to satisfy according to users’ various data requirements (DRs). Existing approaches and tools, such as Damia (Altinel et al., 2007) and iViewer (Han et al., 2013), present a visualization interface to manually generate the data view. These approaches may be effective for a small quantity of DSs; however, as the quantity of DSs grows, it becomes too inefficient for users to generate their desired data view.

To handle these problems, this paper proposes a novel automatic DS composition view generation approach. The approach constructs a data service dependence graph (DSDG) according to the inherent dependences among DSs. It can automatically compose DSs using the DSDG according to user’s DRs and then automatically generate data view. The main contributions of this paper are as follows:

1. We build a DSDG based on the inherent dependences among DSs. This graph describes the whole relations of DSs.
2. We give a DS composition algorithm based on the DSDG. This algorithm can automatically compose DSs according to users’ DRs.
3. We propose a data view generation algorithm and define a set of basic operations including selection, join, projection and set operations to generate data view.
4. We develop a DS composition view generation prototype system, named DSViewer. We evaluate this system with real data sets and evaluated in detail, and demonstrate the system can automatically compose DS and efficiently generate DS view.

The rest of the paper is organized as follows: Section 2 gives the related work on DSs. Section 3 models the relationship between DSs. Section 4 gives a DS composition algorithm. Section 5 shows the DS view generation algorithm. Section 6 presents the DS view system and evaluates the key algorithms in detail, and finally, Section 7 concludes this paper.

2. Related work

DS provides a new effective approach to integrate and share heterogeneous data accessed in cloud environment. Currently, DS has become a hot spot in the field of service computing, especially in current big data era.

Early researches concentrate on DS modeling, DS access and DS applications. Carey et al. (2008) verified that DS cannot only directly access data source but can also integrate into the service-oriented service (SOA) through a standard interface. This technique does not rely on existing applications and can access cross-platform data resources. The technique also makes up the shortcomings of traditional SOA in data access. Yu et al. (2017) proposed a framework that discovers semantic links in printed forms while generating DSs for easy data management and rapid data sharing in enterprise systems. Xu et al. (2018) designed a dynamic DS publishing engine system to process the invocation requests of service. The engine with Restful architecture addresses the problems of data model heterogeneity, data extraction and data synthesis. Badidi and Routaib (2018) introduced a data provisioning framework to help data consumers find high-quality IoT (Internet of Things) data. The framework is based on the data as a service (DaaS) cloud delivery model. It can evaluate the
provision of latent DaaS providers based on the needs of data consumers. Silva et al. (2018) designed a Web Crawler in combination with Middleware for DaaS and SaaS (MI-DAS) to offer a solution for interoperating software as a service (SaaS) and DaaS in the case of data delivery. Li and Zhang (2021) proposed a server-side solution based on FTP protocol to solve the problem that how to provide simple data transmission service in distributed file system. The solution named SPDScheme (Server Protocol Data Scheme) includes an independent service SPDServer based on FTP protocol between the user and distributed file system to ensure high concurrency and scalability of services. Immonen et al. (2018) outlined the kinds of knowledge and services which are required for validating open data in DS ecosystems. Yunkkon and Eui-Nam (2017) proposed a reference model to guarantee DS reliability satisfying various users’ requirements. The model makes user obtain maximum data volume in limited time.

In recent years, DS has been utilized in cloud computing. Vieira et al. (2021) developed the data join system to solve the problem of integrating data from different cloud services. This model is described through the specification model and incorporated into the middleware as a proof of concept. Goga et al. (2018) outlined deployment methods of virtual machines and their applicability to DaaS model in clouds especially virtual machines migration. Psomakelis et al. (2020) designed architecture for a DaaS marketplace hosted in a cloud environment. The architecture includes a storage management engine, a monitoring component and a parsing engine and evaluates the performance and efficiency of applications by strictly regulated data exchange process. Plebani et al. (2018) gave a goal-based modeling approach to achieve effective data movements in fog environments. In fog environments, data is generated at the edge of the network, processed on the cloud and consumed at customer sites. The approach can effectively handle frequent data movement requests. Romdhani et al. (2019) proposed a classification scheme for current trust solutions insisting in open issues in cloud environments. The method gives a general idea of using Service Level Agreement (SLA) to improve multi-cloud data provisioning. Xi et al. (2018) designed a new type of data flow named encryption flow to describe dependencies among different encrypted data objects across multiple services and gives a secure information flow verification theorem.

Some works have been done on DS composition and data view generation. Zhang et al. (2018) proposed a DSDG to automatically compose DSs and generate data views. According to the internal data dependencies, DSs can be converted to DSDG. The visual data view can be got from searching and integrating DSDG. Wang et al. (2018) designed a continuous DS model and a continuous DS composition algorithm in answer to queries across data streams. The model realizes the access and sharing of data streams through DS. Chen et al. (2017) proposed a DS composition sequence generation approach for ad-hoc data query problems. The method on the basis of keywords input by users can find the relevant DSs and generate DS composition sequence as the output. Gu et al. (2018) gave a Web service composition discovery method to find proper DSs and implement a Web service composition that can realize complex and characteristic functions on the service data network. Huo and Zhang (2020) designed a nonlinear service composition method based on the Skyline operator. The method is to quickly find the service composition to solve the problem, and Skyline operator contributes to reducing redundant services. Dara and Emadi (2021) proposed a method to improve the data-driven composition of web services by enriching tags based on tags semantic. The method can provide automatic service composition and automatically search the service compositions for a given query. Liu et al. (2019) proposed a data flow control approach based on dependency analysis for ensuring information flow’s security in cloud composite service. Cai et al. (2019) designed a service composition and optimization model to optimize knowledge service composition under cloud manufacturing. Jia and Wang (2020) introduced a process construction approach for data-oriented users to directly operate on
data views to build a stream service composition process. Faieq et al. (2019) proposed a recommendation-based service composition system targeting smart environments. The system can capture the situation of the users to select appropriate service models to meet their needs. Badidi et al. (2019) introduced an integrated framework for enhancing personalized mobile cloud-services. The framework is based on a composition approach to solve the personalization in mobile cloud-service provisioning. Sellami et al. (2020) proposed an elastic composition algorithm for composing multi-tenant cloud services and performing their elasticity through the proposed service pattern.

Some works have also been performed for data view update and optimization. Zhang et al. (2013) proposed a dynamic update method for nested views based on DSs. This approach uses pointers to create references to tuples in nested views; it uses the update log to record the DS and improve the data freshness of nested views. Zhang et al. (2020) proposed a model based on incremental log data combined view location update to update the data composite view in real time at the minimum cost when the data changes.

Compared with previous works, our proposed approach can organize DSs into a dependence graph by using inherent data dependences. This dependence graph provides a foundation to automatically compose DS according to users’ various requirements. And then, a data view can be generated by interpreting the composed result. It provides a more flexible approach for users to utilize DS in cloud environment.

3. Data service dependence graph construction

We use real data sets to illustrate the DS composition and data view generation. The data sets are extracted from different elevator enterprise departments including design data, sales data and maintenance data. For simplicity, Table 1 gives two only two data sets. One is extracted from design department and another is extracted from maintenance department. This table shows the data structure and their attributes. Generally, an attribute is the abstract characteristic description of an object, and data are the specific values of an attribute. The data dependence is the inherent constraint among data.

Figure 1 shows the attribute dependence graph constructed according to the inner-data dependence of the attributes in Table 1. Each node of the graph represents an attribute, and each arrow of the graph represents dependence between two nodes. For example, the attributes $b$, $c$ and $d$ are dependent on the attribute $a$, and the attribute $a$ is interdependent on the attribute $a'$.

<table>
<thead>
<tr>
<th>Data source</th>
<th>Table name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design depart.</td>
<td>elevator_info</td>
<td>a; b; c; d</td>
</tr>
<tr>
<td></td>
<td>client_info</td>
<td>e; f; g; h</td>
</tr>
<tr>
<td></td>
<td>order_info</td>
<td>i; a'; e'; j</td>
</tr>
<tr>
<td>Maintenance depart.</td>
<td>elevator_info</td>
<td>k; l; m; n</td>
</tr>
<tr>
<td></td>
<td>record_info</td>
<td>o; p; q; r</td>
</tr>
<tr>
<td></td>
<td>component_info</td>
<td>k'; o'; s; t</td>
</tr>
</tbody>
</table>

Table 1. Data structures extracted from two elevator enterprise data sets.

(1) a: Elevator no    (2) b: Elevator model (3) i: Registration no (4) j:Floor number (5) o: Maintenance id (6) k: Registration no

(1) a: Client no     (2) c: Elevator specifications (3) j: Client name (4) m: Building name (5) p: Elevator fault (6) k': Registration no

(2) e: Client name    (3) c': Elevator no     (4) l: Floor number (5) q: Repair time (6) o': Maintenance id

(3) e': Client no     (4) m': Building name   (5) r: Elevator price (6) n: Elevator address

(5) o: Maintenance id (6) s: Maintenance parts
Actually, DSs are obtained from encapsulating a set of attributes. If the encapsulated attributes cannot be further subdivided, the corresponding DS is an atomic data service (ADS). We define the formal definition as follows:

**Definition 1. Atomic data service (ADS).** An accessible and semantically nondividable DS is called an atomic data service. Formally, an ADS is a tuple.

$$\text{ADS} = (ID, \text{Name}, \text{Fields}, \text{Description}, \text{Inputs}, \text{Outputs}, \text{Operations}, \text{Publisher}),$$

where:

1. **ID** represents the identification of the ADS.
2. **Name** represents the name of the ADS.
3. **Fields** represent the encapsulated attributes of the ADS.
4. **Description** represents the semantic information of the ADS.
5. **Inputs** show the multiple input parameters of the ADS.
6. **Outputs** show the execution result of the ADS.
7. **Operations** give the possible operations to the ADS.
8. **Publisher** shows the source of the ADS.

According to the attribute dependence graph, we can extract the ADSs. Table 2 shows the extraction results, which lists all the ADSs extracted from Figure 1. For example, the ADS1 encapsulates the attribute $a$, and accessing ADS1 will return all elevator numbers. Since the ADSs are obtained from the encapsulating attributes, the inherent data dependencies between attributes can be directly mapped onto the dependencies between DSs. We define the DSDG as follows.

**Definition 2. Data service dependence graph (DSDG).** The DSDG is an extended, directed graph that describes the dependencies between ADSs and can be defined as a tuple.

$$\text{DSDG} = (\text{DS}, \text{E}),$$

where

1. **DS** = \{ADS$_1$, ADS$_2$, ..., ADS$_n$\}, in which the ADS$_i$ is an ADS.
2. **E** = \{e$_1$, e$_2$, ..., e$_m$\}, in which e$_i$ = A → ADS$_j$ represents that the ADS$_j$ is dependent on the A (A $\subseteq$ DS).

The DSDG of the DSs in Table 2 is given in Figure 2. It clearly shows the global logical structure of the ADSs and provides a foundation for the DS composition.
4. Data service composition
The ADS encapsulates the attributes of the data and can be accessed through the public interfaces on the web. However, a single ADS only provides simple data and cannot satisfy...
complex demands of users in most situations. It is necessary to compose multiple ADSs according to users’ DRs. We first give a formal definition of the DRs to facilitate the DS composition process.

**Definition 3. Data requirements (DRs).** DRs show the demands of users on the data set. Formally, it can be defined as a tuple.

\[ \text{DR} = (\text{Fields}, \text{Conditions}, \text{Operations}) \]

1. **Fields** are the desired attributes.
2. **Conditions** = \{\langle Field_i, Value_i \rangle | Field_i \in \text{Fields}, Value_i \text{ is a constant} \}, which represent the data restrictions.
3. **Operations** = \{GET, UPDATE, DELETE\}. For example, if a user wants to query the elevator specifications of HZDS, the DR can be defined as follows.

\[ \text{DR} = (\{\text{Client name, Elevator specification}\}, \{< \text{Client name, “HZDS”}>\}, <\text{get}>). \]

The DS composition aims to generate a new complex DS by composing multiple ADSs according to the DR. The composite DS is defined as follows.

**Definition 4. Composite data service (CDS).** CDS is a combination of multiple atomic DSs and can be independently accessed on the web. Formally, CDS is a tuple.

\[ \text{CDS} = (\text{ID, Name, Sub-DSDG, Description, Inputs, Outputs, Operations}) \]

1. **ID** represents the identification of the CDS.
2. **Name** represents the name of the CDS.
3. **Sub-DSDG** is a sub graph of the DSDG.
4. **Description** is the semantic information of the CDS.
5. **Inputs** are the multiple input parameters of the CDS.
6. **Outputs** show the execution result of the CDS.
7. **Operations** are the executable operations including get, update and delete.

We propose an automatic DSs composition algorithm, shown as Algorithm 1. The inputs are the DSDG and the DR, and the output is the CDS. This algorithm will generate a CDS based on the DSDG according to user’s DR. The algorithm selects the field-related ADSs in the fields of the DR and takes the first as the starting ADS. Then, it accesses these ADSs in the DSDG with the breadth-first search strategy and records the prior ADS of the visited ADS until all the fields of the DR are included in the visited queue. A complete access path between the starting ADS and the other ADSs are stored. The ADSs that exist in the complete access paths will be composed into one CDS.

If there are more than one complete access path to these ADSs, there may exist more than one composition result. It is essential to select an optimal CDS from the possible composition results. Since the number of ADSs and the number of attributes in the CDS affect the execution performance of the data view generation, the optimal CDS should have the minimum number of ADSs and attributes. In the algorithm, the DSDG is an unweighted graph; therefore, the breadth-first search strategy can take the path that contains the minimum number ADSs and attributes to generate the optimal CDS. Our later experiments also show that the algorithm can output the optimal composition result.

In addition, the algorithm assumes that all the nodes in the DSDG are connected. If the graph is unconnected, the algorithm will traverse all the subgraphs of the DSDG.
It is assumed that \( r \) is the field number of DRs, \( n \) is the ADS number of the DSDG and \( e \) is the edge number of DSDG. The time complexity of lines 1 to 5 is \( O(n) \), and the time complexity of lines 6 to 33 is \( O(n + e) \). The overall time complexity of the algorithm is \( O(n + e) \).

We use another more complex DR below to illustrate the composition procedure of the algorithm.

\[
\text{DR} = \{\{\text{Client name, Elevator price, Elevator specification, floor number, Building name }, \{\text{<Client name, “HZDS”>}, \text{<building name, “Guangzhi building”>}, \text{<get}>\}.
\]

This algorithm will take the DR and the DSDG of Figure 2 as inputs and search the required ADSs in the DSDG as follows.

\{ADS\_1, ADS\_3, ADS\_5, ADS\_9, ADS\_10, ADS\_11, ADS\_12, ADS\_13, ADS\_14, ADS\_15\}

Then, the algorithm will generate a CDS by composing these ADSs. The gray nodes in Figure 2 are composed into a composite DS.

**Algorithm 1. Data service composition algorithm**

```plaintext
Input: DSDG, DR
Output: CDS
1: function DSComposition(DSDG, DR)
2: fields ← DR
3: for each field in fields do
4: ADSs ← ADS related to field
5: end for
6: InitQueue(queue)
7: EnQueue(queue, ADS0) ▷ The first ADS of ADSs
8: while !QueueEmpty(queue) do
9: ADSi ← DeQueue(queue) ▷ Find all adjacent ADSs of ADSi
10: adj_ADSs ← DSDG ▷ Find all adjacent ADSs of ADS
11: for each ADS in adj_ADSs do
12: if ADS haven’t been visited then
13: EnQueue(queue, ADS) ▷ Get the prior node of cur_ADS
14: pre_ADSs ← ADS, is the prior node of ADS
15: if all ADSs have been visited then
16: break;
17: end if
18: end if
19: end for
20: end while
21: for j = 1 to the size of ADSs do
22: cur_ADS ← ADSj
23: pre_ADS ← pre_ADSs(cur_ADS) ▷ Get the prior node of cur_ADS
24: while cur_ADS is not ADSj do
25: if CDS doesn’t contain cur_ADS then
26: Add cur_ADS into CDS;
27: else
28: break
29: end if
30: cur_ADS ← pre_ADS
31: pre_ADS ← pre_ADSs(cur_ADS)
32: end while
33: end for
34: return CDS
35: end function
```

5. Data view generation
5.1 Basic operations on data view
The performance result of the ADS is represented as a single data view that is a table form. The data view of the CDS can be obtained by merging multiple single data views. We define a
set of basic operations on the data view, including the selection operation, join operation, projection operation and set operation, to generate a full data composition view.

**Definition 5. Selection operation.** The selection operation refers to selecting the tuples that satisfy certain condition from a data view. It can be represented as follows:

\[ \sigma_{\text{condition}}(\text{ADS}) \]

**Definition 6. Join operation.** The joint operation between the ADS1 and ADS2 refers to selecting tuples that satisfy certain condition from the Cartesian product of two data views. It can be represented as follows:

\[ \text{ADS}_1 \bowtie_{X_i \text{ condition } Y_i} \text{ADS}_2 \]

where the \( \bowtie \) is the join operator; \( X_i \) is the field of ADS1; and \( Y_i \) is the field of ADS2.

**Definition 7. Projection operation.** The projection operation refers to selecting the desired fields to construct a new data view. It can be represented as follows:

\[ \Pi_A(\text{ADS}) \]

where the \( A \) represents the desired fields.

In addition to the above operations, there are set operations that include intersection, union and difference operations. These set operations are utilized to merge multiple single data views into a full data composition view. Table 3 lists these basic data view operations.

### 5.2 Data view generation algorithm

Data view of CDS is generated by merging multiple single data views of ADSs. Algorithm 2 gives the data view generation algorithm for a given CDS. Its input is a CDS, which is composed of multiple ADSs and a DR, which specifies the conditions of request data. Its output is a data composition view. The algorithm takes the first field-related ADS in the conditions of DR as the starting node and pushes it into a queue to be visited. This ADS is performed with the field value in the conditions of DR as input. For example, in the condition of \(<\text{Client name}, \text{"HZDS"}>, \) the field value is \("HZDS"\). The condition is stored in a two-dimensional array. The output of the previous ADS is used as input, and the nodes in CDS are sequentially accessed and executed according to the breadth-first strategy. The joint operation is performed on the data view and the current data view. If there are redundant data, the projection operation will be performed, and all unvisited ADSs connected to this ADS are divided. The algorithm continues to visit the ADSs in the queue until it is empty. A local data view is obtained. After that, the algorithm will sequentially access and perform other divided ADSs with the breadth-first strategy, until all ADSs in the CDS are performed.

<table>
<thead>
<tr>
<th>ID</th>
<th>Data view operation</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \sigma_{\text{condition}}(\text{ADS}) )</td>
<td>Select tuples from data view of ADS according to the condition</td>
</tr>
<tr>
<td>2</td>
<td>( \text{ADS}<em>1 \bowtie</em>{\text{condition}} \text{ADS}_2 )</td>
<td>Joint data views of ADS1 and ADS2 according to the condition</td>
</tr>
<tr>
<td>3</td>
<td>( \Pi_{\text{fields}}(\text{ADS}) )</td>
<td>Project specified fields from data view of ADS.</td>
</tr>
<tr>
<td>4</td>
<td>( \text{ADS}_1 \cap \text{ADS}_2 )</td>
<td>Union data views of ADS1 and ADS2.</td>
</tr>
<tr>
<td>5</td>
<td>( \text{ADS}_1 \cup \text{ADS}_2 )</td>
<td>Intersect data views of ADS1 and ADS2.</td>
</tr>
<tr>
<td>6</td>
<td>( \text{ADS}_1 - \text{ADS}_2 )</td>
<td>Data view of ADS1 minus data view of ADS2.</td>
</tr>
</tbody>
</table>

Table 3. Data view operation types
All local data views are joined together in sequence to form a composite view, and then the PROJECT operation and SELECTION operation are respectively performed on the composite view according to all conditions. Finally, the final data view is generated.

Algorithm 2. Data view generation algorithm

The CDS composed in Section 4 is taken as an example. Algorithm 2 selects ADS$\textsubscript{5}$ as the starting node to perform the operation. The ADS$\textsubscript{5}$, ADS$\textsubscript{10}$, ADS$\textsubscript{12}$, ADS$\textsubscript{11}$ and ADS$\textsubscript{9}$ are performed in turn with the breadth-first strategy. A local data view, named VIEW$\textsubscript{1}$, will be generated by joining all performing results of the ADSs.

There may be redundant data after performing ADS$\textsubscript{9}$. To avoid performing ADS$\textsubscript{1}$ with the same value of inputs, all unvisited ADSs connected to ADS$\textsubscript{9}$ are divided, i.e. ADS$\textsubscript{1}$ and ADS$\textsubscript{3}$. The output of ADS$\textsubscript{9}$ is performed with the PROJECTION operation and the result is taken as the input for ADS$\textsubscript{1}$. Then, the algorithm will continue to access the ADSs in the queue to be visited, i.e. ADS$\textsubscript{13}$, ADS$\textsubscript{14}$ and ADS$\textsubscript{15}$. The performing results will be joined with the VIEW$\textsubscript{1}$ in sequence, and the data view named VIEW$\textsubscript{2}$ will be generated. When the queue is empty, the
ADSs that were previously divided are accessed. The ADS₁ and ADS₃ are pushed into the queue and performed sequentially, each performing result of the ADS is joined with another new data view named VIEW₃. The field-related ADSs of VIEW₁, VIEW₂ and VIEW₃ are drawn in Figure 3.

Until now, all ADSs in the CDS are performed, and the VIEW₂ and VIEW₃ are performed with set operation to generate a composition view. The composite data view is performed with the projection operation and selection operation according to the fields and conditions of the DR, respectively, to generate the final composition view.

6. Experimental results
Since there are no public benchmarks, we utilize real cross-organization elevator data to evaluate the proposed approach. There are elevator design data, elevator sales data, elevator fault data, elevator customer data, elevator manufacturing data and elevator maintenance data, which are extracted from different elevator enterprises.

6.1 Prototype system
We have developed a service-based data view system, called DSViewer, to implement the data integration and data sharing. Currently, the main functions of the system include DSs extraction, DS composition, composition view generation and DS management. The system can automatically generate data composition views for users.

The DSs extraction can automatically extract ADSs from the data source and establish dependencies among the ADSs. Figure 4 shows the ADSs and their DSDG of the elevator data. This figure intuitively represents the relationships between ADSs. In the system, all ADSs are encapsulated into the RESTful services that can be accessed on the web.

The DS composition can generate a CDS by composing the ADSs according to the user’s DRs. Figure 5 shows the DS composition interfaces. Figure 5(a) shows the DR definition interface and Figure 5(b) shows the composition results shown with a sub-DSDG. The CDS can be stored and directly accessed on the web.

The data view generation can perform the CDS and output a data composition view. Users can conveniently select one CDS that satisfies their demands and define the query conditions. Figure 6 shows the data view generation interface, where the top is utilized to define the query conditions, and the bottom is utilized to output the data view.

6.2 Performance evaluation
In this subsection, we will evaluate two key algorithms adopted in the SDCViewer: the DS composition algorithm and the composition view generation algorithm. The experimental
Figure 4. Data service dependence graph of ADSs
Sharing large data collections

Figure 5. Data service composition interfaces
hardware is a 2.50 GHz 8-core CPU, 16 GB RAM, and 290 GB disk storage. The operation system is a 64-bit Ubuntu 16.04. All algorithms are implemented with the JAVA programming language.

6.2.1 Data service composition performance. Since the attributes of the conditions are a sub set of attributes of the fields, the conditions of the DR do not affect the composition. We design the two kinds of data sets shown in Table 4.

(1) Data set 1 keeps the attribute number of fields unchanged and varies the total ADS number.

(2) Data set 2 keeps the total ADS number unchanged and varies the attribute number of the fields.

Each test is given a random attribute list, and the average of ten test results is taken as the experimental result.

We first evaluate the composition performance that aims to show the overall time consumed by the composition algorithm. Figure 7 shows the composition performance with different ADS numbers and different attribute numbers. Figure 7(a) shows the overall time consumed to complete the composition by varying the ADS number, where the X axis represents the ADS number and the Y axis represents the overall time. It can be seen that the overall time is increased with the increasing ADS number. The reason is that large ADSs affect the scope of the DS composition. Figure 7(b) shows the overall time required to complete

<table>
<thead>
<tr>
<th>Experimental data sets for the data service composition algorithm</th>
<th>Attribute number of fields</th>
<th>Total ADS number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set 1</td>
<td>6</td>
<td>100, 200, 400, 600, 800</td>
</tr>
<tr>
<td>Data set 2</td>
<td>2, 4, 6, 8, 10</td>
<td>800</td>
</tr>
</tbody>
</table>

Figure 6. Data view generation interface
the composition with different attribute numbers. This figure represents similar patterns as Figure 7(a). That is, the total time also increased with the increasing attribute number. More attributes in the fields will require more ADSs to compose the CDS.

Then, we evaluate the composition quality that aims to check whether the composition result is optimal. As discussed earlier, there may exist more than one composition result for a given DR. However, the ADS and attribute numbers may be different. The optimal composition results should have the minimum ADS number and attribute number. Table 5 gives the statistics of the composition results with different attribute numbers. When the attribute number of the fields is two, there are three different composition results that all meet the DR. The first CDS includes four ADSs and four attributes; the second CDS includes four ADSs and five attributes; and the third CDS includes eight ADSs and eight attributes. Since the first CDS has the minimum ADS and attribute numbers, it is the optimal composition result. In addition, the experimental results show that large attribute numbers of the fields will have fewer composition results. For example, when the attribute number of the fields is ten, there is only one composition result.

6.2.2 Data view generation performance. We further evaluate the data view generation algorithm. To evaluate this algorithm, we design three other kinds of data sets, as shown in Table 6.

<table>
<thead>
<tr>
<th>Attribute number of fields</th>
<th>Composition results</th>
<th>ADS number</th>
<th>Attribute number</th>
<th>Optimal CDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>4</td>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>8</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6</td>
<td>6</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>8</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>9</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
<td>7</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
<td>N</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>9</td>
<td>9</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td>N</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>12</td>
<td>12</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 5. Experimental statistics of the composition quality
(1) Data set 1 keeps the attribute number of the fields and conditions unchanged and varies the tuple number.

(2) Data set 2 keeps the tuple number and the attribute number of the conditions unchanged and varies the attribute number of the fields.

(3) Data set 3 keeps the tuple number and the attribute number of fields unchanged and varies the attribute number of the conditions.

Figure 8 shows the performance of the composition view generation algorithm with different parameters. Figure 8(a) shows the performance with different tuple numbers, where the $X$ axis represents the tuple number, and the $Y$ axis represents the time to generate a data view. It can be seen that the performance is decreased with the increasing tuple number. The reason is

<table>
<thead>
<tr>
<th>Data set</th>
<th>Attribute number of fields</th>
<th>Attribute number of conditions</th>
<th>Tuple number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set 1</td>
<td>6</td>
<td>1</td>
<td>200, 400, 600, 800, 1000</td>
</tr>
<tr>
<td>Data set 2</td>
<td>2, 4, 6, 8, 10</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>Data set 3</td>
<td>6</td>
<td>1, 2, 4, 6</td>
<td>200</td>
</tr>
</tbody>
</table>
that more tuples will require more SET operations and greater communication time, therefore consuming more time. Figure 8(b) shows the performance with different attribute numbers of the fields, where the $X$ axis represents the attribute number of the fields, and the $Y$ axis represents the time to generate a data view. It can be seen that the performance also decreases with the increasing attribute number of the fields. The reason is that more attributes will require more JOINT operations and, therefore, will consume more time. Figure 8(c) shows the performance with different attribute numbers of the conditions, where the $X$ axis represents the attribute number of the conditions, and the $Y$ axis represents the time to generate a data view. It can be seen that the performance also decreases with the increasing attribute number of the conditions. The reason is that more attributes of the conditions will require more SELECTION operations and SET operations and, therefore, will consume more time.

We also evaluate the generation accuracy rate of the algorithm to check whether the data view meets a given DR. The execution process of the algorithm reveals that the attributes contained in the data view match the attributes contained in the requirements of the DR, and the performing results of the CDS satisfy the conditions of the DR. This result indicates that the final data view can satisfy the DR accurately, and our actual experimental results also demonstrate this conclusion. Therefore, the generation accuracy rate of the algorithm is 100%.

7. Conclusions
To automatically generate data view on demand from a large number of DSs, we presented an automatic DS composition and view generation approach. A DSDG is built according to the inherent dependence and it presents a global perspective on the relationship of DSs. Based on the DSDG, the DSs can be automatically composed and then data view can be automatically generated. We have developed a DS view generation system (DSViewer) that enables DS extraction, DS composition and data view generation. This system provides an effective tool to integrate heterogeneous cross-organization data. We have evaluated the system and key algorithms and showed the correctness and effectiveness to generate a desired data view for users. Our future work will concentrate on the real-time data view update and unstructured DS integration.

References


**Corresponding author**

Yuanming Zhang can be contacted at: zym@zjut.edu.cn

For instructions on how to order reprints of this article, please visit our website: [www.emeraldgrouppublishing.com/licensing/reprints.htm](http://www.emeraldgrouppublishing.com/licensing/reprints.htm)

Or contact us for further details: permissions@emeraldinsight.com