Received 20 January 2022
Revised 6 March 2022
Accepted 6 March 2022

# 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-origination 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-origination 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



*Funding:* The work is being supported by the National Natural Science Foundation of China under Grant NO. 61976193.

Journal of Intelligent Manufacturing and Special Equipment
Vol. 3 No. 1, 2022
pp. 48-66
Emerald Publishing Limited
e-ISSN: 2633-660X
p-ISSN: 2633-6596
DOI 10.1108/JIMSE-01-2022-0003

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 deliver. 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. Yunkon 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 uses' 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'*.

| Data source | Table name | Attributes | |
|---|---|---|---|
| Design depart. | elevator_info | a; b; c; d | |
| | client_info | e; f; g; h | |
| | order_info | i; a′; e'; j | |
| Maintenance depart. | elevator_info | k; l; m; n | |
| | record_info | o; p; q; r | |
| | component_info | k′; o'; s; t | |
| (1) a: Elevator no | b: Elevator model | c: Elevator specifications | d: Elevator interior |
| (2) e: Client no | f: Client name | g: Client address | h: Client contact |
| (3) i: Registration no | a′: Elevator no | e′: Client no | j: Elevator price |
| (4) k: Registration no | l: Floor number | m: Building name | n: Elevator address |
| (5) o: Maintenance id | p: Elevator fault | q: Repair time | r: Maintenance time |
| (6) k': Registration no | o′: Maintenance id | s: Maintenance parts | t: Maintenance price |

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

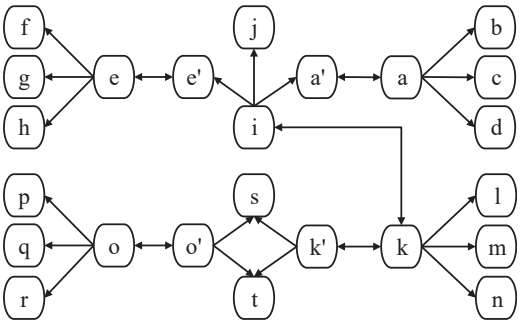



**Figure 1.**
Attribute dependence graph

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.
ADS = (ID, Name, Fields, Description, Inputs, Outputs, Operations, 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 $ADS_1$ encapsulates the attribute $a$, and accessing $ADS_1$ 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.
DSDG = (DS, E), where

(1) $DS = \{ADS_1, ADS_2, \ldots ADS_n\}$, *in which the* $ADS_i$ *is an ADS.*

(2) $E = \{e_1, e_2, \ldots e_m\}$, *in which* $e_i = A \rightarrow ADS_j$ *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.

| ID | Name | Fields | Description | Input | Output | Operation | Publisher |
|---|---|---|---|---|---|---|---|
| 01 | $ADS_1$ | {a} | Query elevator no | Elevator no | Elevator no | Get | Design depart. |
| 02 | $ADS_2$ | {a, b} | Query elevator no, elevator model | Elevator no or elevator model | Elevator no and elevator model | Get | Design depart. |
| 03 | $ADS_3$ | {a, c} | Query elevator no, elevator specifications | Elevator no or elevator specifications | Elevator no and elevator specifications | Get | Design depart. |
| 04 | $ADS_4$ | {a, d} | Query elevator no, elevator interior | Elevator no or elevator interior | Elevator no and elevator interior | Get | Design depart. |
| 05 | $ADS_5$ | {e} | Query client no | Client no | Client no | Get | Design depart. |
| 06 | $ADS_6$ | {e, f} | Query client no, client name | Client no or client name | Client no and client name | Get | Design depart. |
| 07 | $ADS_7$ | {e, g} | Query client no, client address | Client no or client address | Client no and client address | Get | Design depart. |
| 08 | $ADS_8$ | {e, h} | Query client no, client contact | Query client no or client contact | Query client no and client contact | Get | Design depart. |
| 09 | $ADS_9$ | {i, a′} | Query registration no, elevator no | Registration no or elevator no | Registration no and elevator no | Get | Design depart. |
| 10 | $ADS_{10}$ | {i, e′} | Query registration no, client no | Registration no or client no | Registration no and client no | Get | Design depart. |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 24 | $ADS_{24}$ | {k, o, t} | Query registration no, maintenance id, maintenance parts, maintenance price | Registration no, maintenance id or maintenance price | Registration no, maintenance id and maintenance price | Get | Maintenance depart. |

**Table 2.**
Data structures extracted from elevator enterprise database

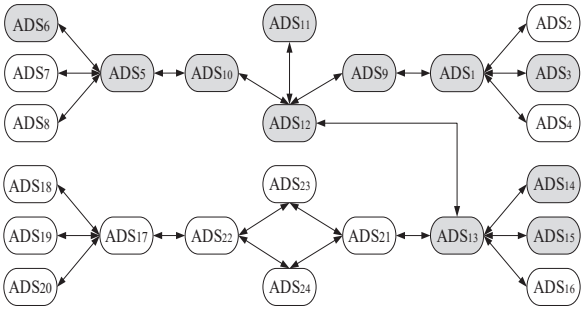



**Figure 2.**
Data service dependence graph

## 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.
DR = (Fields, Conditions, Operations), where

(1) *Fields are the desired attributes.*

(2) *Conditions* = $\{<Field_i, Value_i> | Field_i \in Fields, Value_i$ *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.
DR = ({Client name, Elevator specification}, {< Client name, "HZDS">}, <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.
CDS = (ID, Name, Sub-DSDG, Description, Inputs, Outputs, Operations), where

(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, $\boldsymbol{n}$ is the ADS number of the DSDG and $e$ is the edge number of DSDG. The time complexity of lines 1 to 5 is $\boldsymbol{O(n)}$, and the time complexity of lines 6 to 33 is $\boldsymbol{O(n+e)}$. The overall time complexity of the algorithm is $\boldsymbol{O(n+e)}$.

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

DR = ({Client name, Elevator price, Elevator specification, floor number, Building name }, {<Client name, "HZDS">, <building name, "Guangzhi building">}, <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.

{$ADS_1$, $ADS_3$, $ADS_5$, $ADS_6$, $ADS_9$, $ADS_{10}$, $ADS_{11}$, $ADS_{12}$, $ADS_{13}$, $ADS_{14}$, $ADS_{15}$}

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

```
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)
10:         adj_ADSs ← DSDG     ▷Find all adjacent ADSs of ADSi
11:         for each ADS in adj_ADSs do
12:             if ADS haven't been visited then
13:                 EnQueue(queue, ADS)
14:                 pre_ADSs ← ADSi 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 → 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 ADS1 do
25:             if CDS dosen'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 $\text{ADS}_1$ and $\text{ADS}_2$ refers to selecting tuples that satisfy certain condition from the Cartesian product of two data views. It can be represented as follows:

$$\text{ADS}_1 \underset{X_i \text{ condition } Y_i}{\bowtie} \text{ADS}_2$$

where the $\bowtie$ is the join operator; $X_i$ is the field of $\text{ADS}_1$; and $Y_i$ is the field of $\text{ADS}_2$.

*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 <*Client name, "HZDS"*>, the filed 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.

| ID | Data view operation | Function description |
|---|---|---|
| 1 | $\sigma_{\text{condition}}(\text{ADS})$ | Select tuples from data view of ADS according to the condition |
| 2 | $\text{ADS}_1 \underset{\text{condition}}{\bowtie} \text{ADS}_2$ | Joint data views of $\text{ADS}_1$ and $\text{ADS}_2$ according to the condition |
| 3 | $\prod\text{fields(ADS)}$ | Project specified fields from data view of ADS. |
| 4 | $\text{ADS}_1 \cap \text{ADS}_2$ | Union data views of $\text{ADS}_1$ and $\text{ADS}_2$ |
| 5 | $\text{ADS}_1 \cup \text{ADS}_2$ | Intersect data views of $\text{ADS}_1$ and $\text{ADS}_2$ |
| 6 | $\text{ADS}_1 - \text{ADS}_2$ | Data view of $\text{ADS}_1$ minus data view of $\text{ADS}_2$ |

**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

```
Input: CDS, DR
Output: DCV
 1: function DCVGeneration(CDS, DR)
 2:     conditions ← DR
 3:     for each condition in conditions do
 4:         ADSs ← ADS related to condition
 5:         condition_values ← condition
 6:     end for
 7:     InitQueue(queue1)
 8:     EnQueue(queue1, ADS1)                          ▷The first ADS of ADSs
 9:     while !QueueEmpty(queue1) do
10:         ADSi ← DeQueue(queue1)
11:         InitQueue(queue2)
12:         EnQueue(queue2, ADSi)
13:         Add ADSi and input value of ADSi into sub_dataView
14:         while !QueueEmpty(queue2) do
15:             ADSj ← DeQueue(queue2)
16:             adj_ADSs ← CDS                         ▷Find all adjacent ADSs of ADSj
17:             for each ADS in adj_ADSs do
18:                 if ADS haven't been visited then
19:                     result ← Execute(ADS)
20:                     Add ADS and result into sub_dataView
21:                     if the condition may create data redundancy then
22:                         ADSk ← The next adjacent node of ADS
23:                         EnQueue(queue1, ADSk)
24:                     else
25:                         EnQueue(queue2, ADS)
26:                     end if
27:                 end if
28:             end for
29:         end while
30:         Add ADSi and sub_dataView into dataView
31:     end while
32:     if the size of dataView > 0 then
33:         for j = 0 → the size of dataView do
34:             DCV ← Join(DCV, dataView[i])
35:         end for
36:     end if
37:     DCV ← Projection(DCV, DR.fields );
38:     DCV ← Selection(DCV, DR.conditions );
39:     return DCV
40: end function
```

The CDS composed in Section 4 is taken as an example. Algorithm 2 selects $ADS_6$ as the starting node to perform the operation. The $ADS_5$, $ADS_{10}$, $ADS_{12}$, $ADS_{11}$ and $ADS_9$ are performed in turn with the bread-first strategy. A local data view, named $VIEW_1$, will be generated by joining all performing results of the ADSs.

There may be redundant data after performing $ADS_9$. To avoid performing $ADS_1$ with the same value of inputs, all unvisited ADSs connected to $ADS_9$ are divided, i.e. $ADS_1$ and $ADS_3$. The output of $ADS_9$ is performed with the *PROJECTION* operation and the result is taken as the input for $ADS_1$. Then, the algorithm will continue to access the ADSs in the queue to be visited, i.e. $ADS_{13}$, $ADS_{14}$ and $ADS_{15}$. The performing results will be joined with the $VIEW_1$ in sequence, and the data view named $VIEW_2$ will be generated. When the queue is empty, the

ADSs that were previously divided are accessed. The $ADS_1$ and $ADS_3$ are pushed into the queue and performed sequentially, each performing result of the ADS is joined with another new data view named $VIEW_3$. The field-related ADSs of $VIEW_1$, $VIEW_2$ and $VIEW_3$ are drawn in Figure 3.

Until now, all ADSs in the CDS are performed, and the $VIEW_2$ and $VIEW_3$ 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

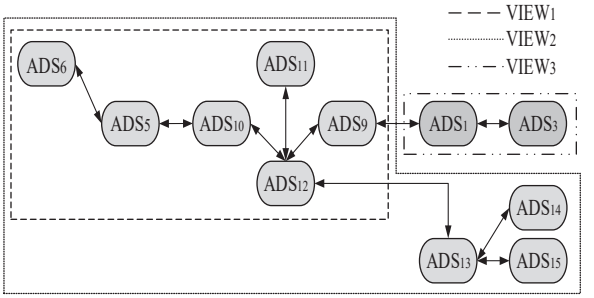



**Figure 3.**
The sub-DSDG of CDS

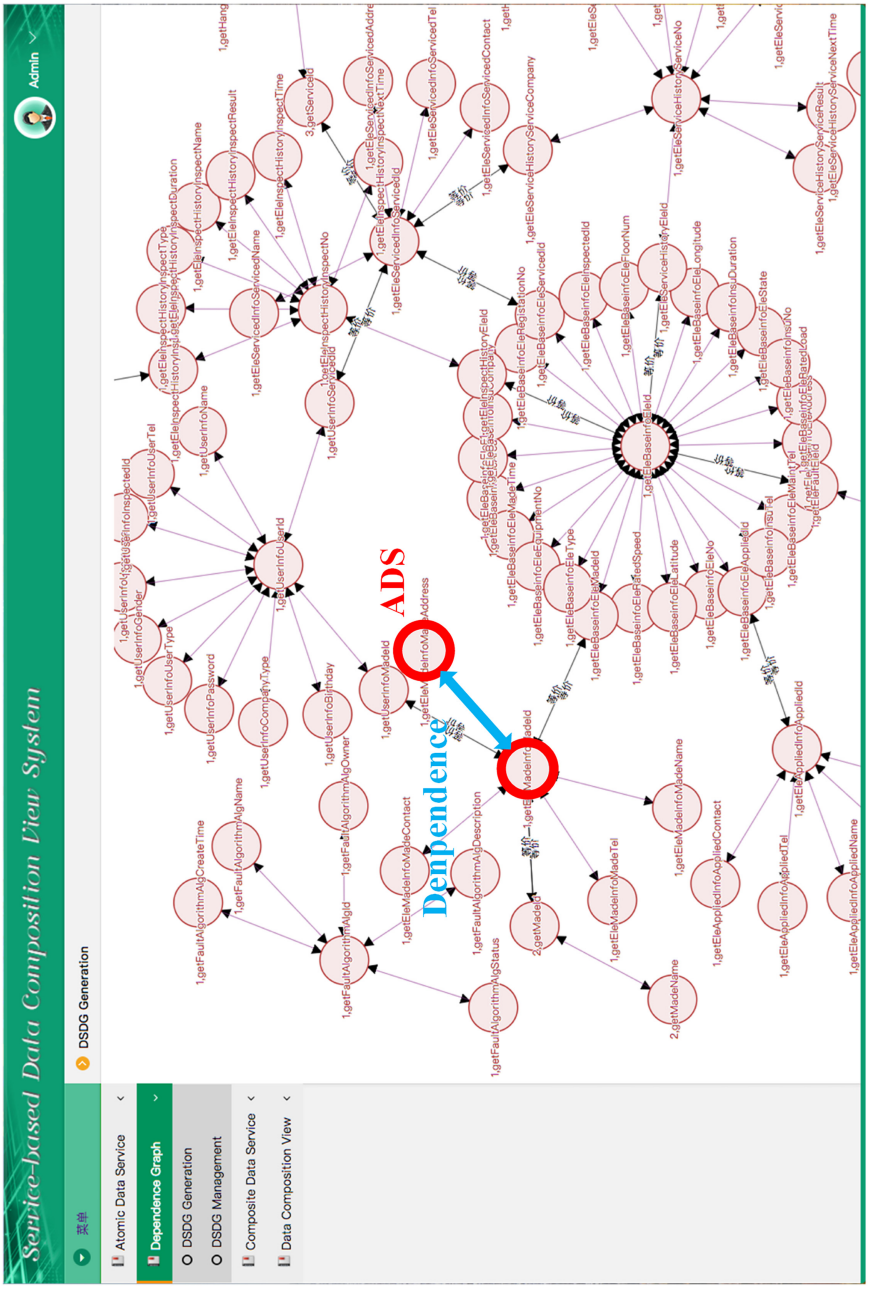



**Figure 4.**
Data service dependence graph of ADSs

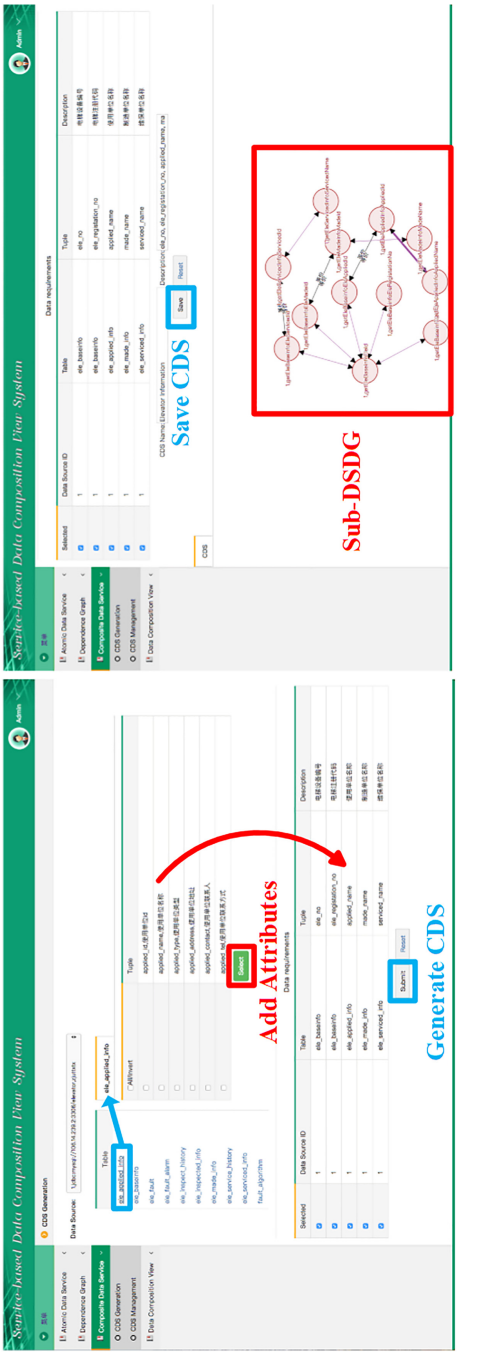



**Figure 5.**
Data service composition interfaces

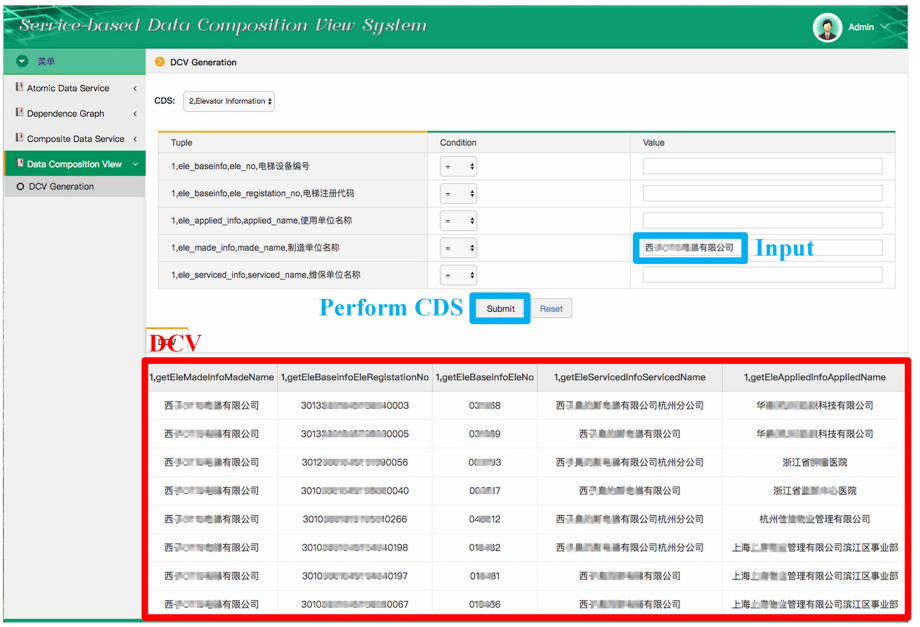



**Figure 6.**
Data view generation interface

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.

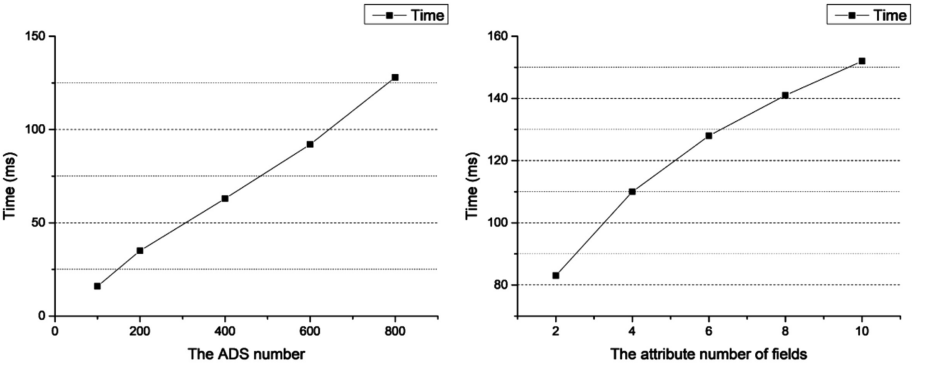
(2) Data set 2 keeps the total ADS number unchanged and varies the attribute number of the fields.

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

| Data set type | Attribute number of fields | Total ADS number |
|---|---|---|
| Data set 1 | 6 | 100, 200, 400, 600, 800 |
| Data set 2 | 2, 4, 6, 8, 10 | 800 |

**Table 4.**
Experimental data sets for the data service composition algorithm




Different ADS number
**(a)**

Different attribute number of fields
**(b)**

**Figure 7.**
Performance of data service composition algorithm

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 5.**
Experimental statistics of the composition quality

| Attribute number of fields | Composition results | | Optimal CDS |
|---|---|---|---|
| | ADS number | Attribute number | |
| 2 | *4* | *4* | *Y* |
| | 4 | 5 | N |
| | 8 | 8 | N |
| 4 | *6* | *6* | *Y* |
| | 8 | 8 | N |
| | 8 | 9 | N |
| 6 | *7* | *7* | *Y* |
| | 10 | 10 | N |
| 8 | *9* | *9* | *Y* |
| | 12 | 12 | N |
| 10 | *12* | *12* | *Y* |

(1) Data set 1 keeps the attribute number of the fields and conditions unchanged and varies the tuple number.

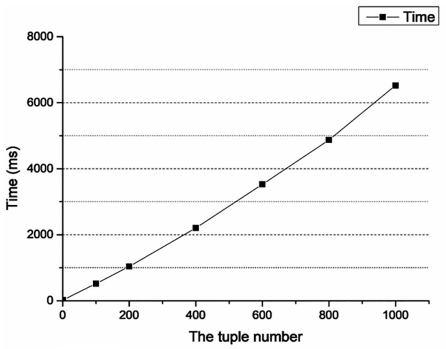
(2) Data set 2 keeps the tuple number and the attribute number of the conditions unchanged and varies the attribute number of the fields.

(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

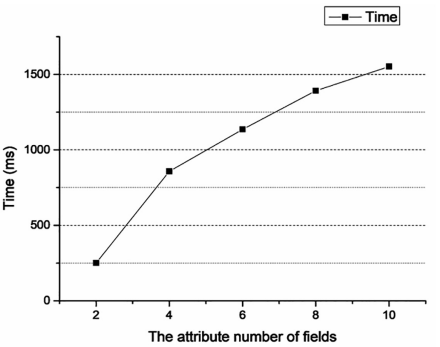
| Data set | Attribute number of fields | Attribute number of conditions | Tuple number |
|---|---|---|---|
| Data set 1 | 6 | 1 | 200, 400, 600, 800, 1000 |
| Data set 2 | 2, 4, 6, 8, 10 | 1 | 200 |
| Data set 3 | 6 | 1, 2, 4, 6 | 200 |

**Table 6.**
Experimental statistics of the composition quality



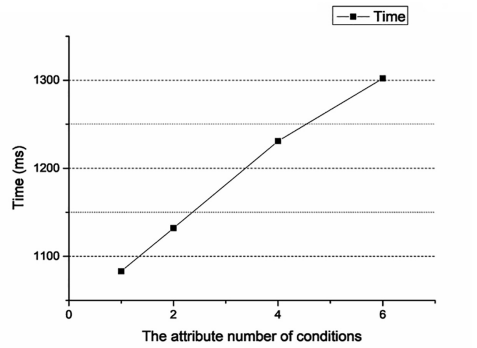

Different tuple number

**(a)**




Different attribute number of fields

**(b)**




Different attribute number of conditions

**(c)**

**Figure 8.**
Performance of the data service composition algorithm

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

Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Ng, Y.H., Simmen, D. and Singh, A. (2007), "Damia: a data mashup fabric for intranet applications", *International Conference on Very Large Data Bases*, pp. 1370-1373.

Badidi, E., Atif, Y., Sheng, Q. and Maheswaran, M. (2019), "On personalized cloud service provisioning for mobile users using adaptive and context-aware service composition", *Computing*, Vol. 101 No. 4, pp. 291-318, doi: 10.1007/s00607-018-0631-8.

Badidi, E. and Routaib, H. (2018), "A DaaS based framework for IoT data provisioning", *Advances in Intelligent Systems and Computing*, Vol. 661, pp. 369-379, doi: 10.1007/978-3-3-319-67618-0_34.

Cai, A.J., Guo, Z.X., Guo, S.H., Cai, Y. and Xue, X. (2019), "Optimization strategy of knowledge service composition in cloud manufacturing environment", *Computer Integrated Manufacturing Systems*, Vol. 25 No. 2, pp. 421-430, doi: 10.13196/j.cims.2019.02.015.

Carey, M., Reveliotis, P., Thatte, S. and Westmann, T. (2008), "Data service modeling in the AquaLogic data services platform", *IEEE Congress on Services*, IEEE, Honolulu, pp. 78-80.

Chen, X., Han, Y., Wen, Y., Zhang, F. and Liu, W. (2017), "A keyword-driven data service composition sequence generation approach on ad-hoc data query", *2017 IEEE International Conference on Web Services,* IEEE, Honolulu, pp. 874-877.

Dara, N. and Emadi, S. (2021), "Enriching web services tags to improve data-driven web services composition", *Journal of Web Engineer*, Vol. 20 No. 2, pp. 327-358, doi: 10.13052/jwe1540-9589.2025.

Faieq, S., Front, A., Saidi, R., Ghazi, H.E. and Rahmani, M.D. (2019), "A context-aware recommendation-based system for service composition in smart environments", *Service Oriented Computing and Applications*, Vol. 13 No. 4, pp. 341-355, doi: 10.1007/s11761-019-00277-7.

Goga, K., Xhafa, F. and Terzo, O. (2018), "VM deployment methods for DaaS model in clouds", *6th International Conference on Emerging Internet, Data and Web Technologies*, Springer, Berlin, pp. 371-382.

Gu, Q., Cao, J. and Yang, X. (2018), "A web services composition discovery approach based on service data network", *IEEE International Conference on Progress in Informatics and Computing,* IEEE, Suzhou, pp. 344-350.

Han, Y., Wang, G., Ji, G. and Zhang, P. (2013), "Situational data integration with data services and nested table", *Service Oriented Computing and Applications*, Vol. 7 No. 2, pp. 129-150.

Huo, Y. and Zhang, J.D. (2020), "A nonlinear service composition method based on the Skyline operator", *Journal of Systems Engineering and Electronics*, Vol. 31 No. 4, pp. 743-750, doi: 10.23919/JSEE.2020.000049.

Immonen, A., Ovaska, E. and Paaso, T. (2018), "Towards certified open data in digital service ecosystems", *Software Quality Journal*, Vol. 26 No. 4, pp. 1257-1297, doi: 10.1007/s11219-017-9378-2.

Jia, S.J. and Wang, J. (2020), "Construction method of user-centered stream data processing application", *Computer Technology and Development*, Vol. 30 No. 12, pp. 45-50, doi: 10.3969/j.issn.1673-629X.2020.12.008.

Li, Y. and Zhang, H.M. (2021), "Data service solution based on FTP protocol", *Computer Systems and Applications*, Vol. 30 No. 1, pp. 63-69, doi: 10.15888/j.cnki.csa.007736.

Liu, M.C., Wang, N. and Zhou, N. (2019), "Dependency analysis based cloud composition service information flow control mechanism", *Computer Science*, Vol. 46 No. 4, pp. 189-196, doi: 10.11896/j.issn.1002-137X.2019.04.030.

Malki, A., Barhamgi, M., Benslimane, S.M., Benslimane, D. and Malki, M. (2014), "Composing data services with uncertain semantics", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 27 No. 4, pp. 936-949.

Plebani, P., Salnitri, M. and Vitali, M. (2018), "Fog computing and data as a service: a goal-based modeling approach to enable effective data movements", *30th International Conference on Advanced Information Systems Engineering,* Springer, Tallinn, pp. 203-219.

Psomakelis, E., Nikolakopoulos, A., Marinakis, A., Psychas, A., Moulos, V., Varvarigou, T. and Christou, A. (2020), "A scalable and semantic data as a service marketplace for enhancing cloud-based applications", *Future Internet*, Vol. 12 No. 5, pp. 77-97, doi: 10.3390/fi12050077.

Romdhani, S., Bennani, N., Ghedira-Guegan, C. and Vargas-Solar, G. (2019), "Trusted data integration in service environments: a systematic mapping", *International Conference on Service-Oriented Computing,* Springer, Toulouse, pp. 237-242.

Sellami, W., Hadj Kacem, H. and Hadj Kacem, A. (2020), "Dynamic provisioning of service composition in a multi-tenant SaaS environment", *Journal of Network and Systems Management*, Vol. 28 No. 4, pp. 367-397, doi: 10.1007/s10922-019-09510-2.

Silva, N., Ribeiro, E.L.F. and Claro, D.B. (2018), "DaaS repository through MIDAS web Crawler", *14th Brazilian Symposium on Information Systems, Association for Computing Machinery*, pp. 246-253.

Vieira, M.A., Ribeiro, E.L.F., Claro, D.B. and Mane, B. (2021), "Integration model between heterogeneous data services in a cloud", *Journal of Universal Computer Science*, Vol. 27 No. 4, pp. 387-412, doi: 10.3897/jucs.67046.

Wang, G.L., Han, Y.B., Zhang, Z.M. and Zhu, M.L. (2017), "Cloud-based integration and service of streaming data", *Chinese Journal of Computers*.

Wang, G.L., Zuo, X., Hesenius, M., Xu, Y., Han, Y. and Gruhn, V. (2018), "A data services composition approach for continuous query on data streams", *2nd International Joint Conference on Asia-Pacific Web*, Springer, Cham, pp. 106-120.

Xi, N., Ma, J., Sun, C., Lu, D. and Shen, Y. (2018), "Information flow control on encrypted data for service composition among multiple clouds", *Distributed Parallel Databases*, Vol. 36 No. 6, pp. 511-527, doi: 10.1007/s10619-018-7228-2.

Xie, J. and Xiao, L. (2014), "Research on data integration based on virtual view and its application", *Engineering Journal of Wuhan University*, Vol. 47 No. 2, pp. 281-285.

Xu, X.S., Wang, Z., Guo, D.D., Wang, Y. and Kang, Y. (2018), "Design and implementation of a dynamic data service publishing engine", *Computer Applications and Software*, Vol. 35 No. 7, pp. 126-130, doi: 10.3969/j.issn.1000-386x.2018.07.022.

Yu, H., Cai, H., Zhou, J. and Jiang, L. (2017), "Data service generation framework from heterogeneous printed forms using semantic link discovery", *Future Generation*, Vol. 79 No. 2, pp. 514-527, doi: 10.1016/j.future.2017.09.059.

Yunkon, K. and Eui-Nam, H. (2017), "Study on user customized data service model for improving data service reliability", *11th International Conference on Ubiquitous Information Management and Communication*, Association for Computing Machinery, New York, pp. 1-8.

Zhang, P., Han, Y.B. and Wang, G.L. (2013), "Implementing dynamic nested view update based on data service", *Chinese Journal of Computers*, Vol. 36 No. 2, pp. 226-237, doi: 10.3724/SP.J.1016.2013.00226.

Zhang, Y.M., Li, M.N., Huang, L.Y., Lu, J.W. and Xiao, G. (2020), "Data composition view positioning update approach with incremental logs", *Computer Science*, Vol. 47 No. 6, pp. 85-91, doi: 10.11896/jsjkx.190500085.

Zhang, Y.M., Ye, C.L. and Huang, L.Y. (2018), "Research on data service dependency graph model and automatic composition", *Journal of Chinese Computer Systems*, Vol. 39 No. 3, pp. 450-456, doi: 10.3969/j.issn.1000-1220.2018.03.009.

Zorrilla, M. and García-Saiz, D. (2013), "A service oriented architecture to provide data mining services for non-expert data miners", *Decision Support Systems*, Vol. 55 No. 1, pp. 399-411.

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