An Automatic Data Service Generation Approach for Cross-origin Datasets

Yuanming Zhang(✉), Langyou Huang, Jiawei Lu, and Gang Xiao

Zhejiang University of Technology, Hangzhou, China
{zym, 2111612010, viivan, xg}@zjut.edu.cn

Abstract. As a unified data access model, data service has become a promising technique to integrate and share heterogeneous datasets. In order to publish overwhelming data on the web, it is a key to automatically extract and encapsulate data services from various datasets in cloud environment. In this paper, a novel data service generation approach for cross-origin datasets is proposed. An attribute dependency graph (ADG) is constructed by using inherent data dependency. Based on the ADG, an automatic data service extraction algorithm is implemented. The extracted atomic data services are further organized into another representation named data service dependency graph (DSDG). Then, a data service encapsulation framework, which includes an entity layer, a data access object layer and a service layer, is designed. Via a flexible RESTful service template, this framework can automatically encapsulate the extracted data services into the RESTful services which can be accessed by the exposed interfaces. In addition, a data service generation system has been developed. Experimental results show that the system has high efficiency and good quality for data service generation.

Keywords: Data service · Service extraction · Service encapsulation · Service dependency graph · Cross-origin datasets

1 Introduction

Large numbers of datasets are increasingly being published by organizations, such as national data[1], which allow the public to access on the web. Generally, data from different organizations tends to be in different formats and organized differently. To enable better utilization of existing data resources and to reduce duplication of data collection, it is necessary to integrate data from distributed and autonomous datasets while maintaining data integrity and consistency. Then, a unified data access interface for upper-layer applications is required.

The data integration methods based on database federation have good scalability but low query efficiency. Data reproduction methods have a short processing time when the datasets are widely distributed and the network delay is large, but at a high cost,

[1] http://data.stats.gov.cn.

such as data warehouse [8]. Furthermore, the overwhelming data in big data era [6] become too large and complex to be effectively processed by these traditional approaches.

With the rapid development of service computing, the content and scope of services are expanded dramatically. Not only are various functions of software encapsulated into the services, named web service (WS) [1], but diverse data produced from software are also encapsulated into services, called data service (DS) [5]. Data service, deployed on the web, shields heterogeneous datasets through a set of access interfaces and provides a unified model for cross-origin data access, data sharing and data analysis. The service-based integration approach is based on XML and HTTP, which is widely adopted by the industry as standard protocols, and can overcome the defects of traditional data integration. Many companies offer the data service interfaces that are exposed to get easier data access, such as the Google, Twitter and Facebook APIs. However, these conventional data service interfaces fail to support responsive and comprehensive data retrieval [18].

It would be beneficial for the service-based integration to automatically generate the data services and satisfy complex data requirements. Several key issues should be handled. First, automatic data service extraction is the initial step. The current existing data services are designed to answer the specific data requirements, which is grounded on the underlying database schema and pre-assembled index [7]. The service granularity should also be considered to improve the reusability and flexibility for decoupling the data services from the clients, and data services can be further composed. Second, automatic data service encapsulation is necessary. There is no standard mechanism to achieve implementation details of data services. The existing approach [17] requires developers to manually implement data services, which is labor-intensive and repetitive and may results in inconsistent style and non-standard service internal implementation. Furthermore, with the enormous explosion of datasets, it is practically impossible to manually encapsulate a large number of data services. Third, an appropriate data service modeling technique is required. Some work [10] has been done to integrate multiple datasets by using data formal semantics and ontology-based concepts based on data service. The inherent data dependencies can be mapped onto the dependencies among services, and the data services are further organized into a service dependency graph for automatic service composition.

In this paper, we present an automatic data service generation approach. Our main contributions include:

1. We propose an automatic data service extraction approach and organize the data services into a dependency graph according to their inherent dependencies.
2. We design a data service encapsulation framework that can automatically generate RESTful services with a flexible template.
3. We develop a system which has functions of data service extraction, encapsulation and management. This system has high efficiency and good quality with the actual datasets.

The remainder of this paper is structured as follows. Section 2 reviews the related work on data services. Section 3 introduces an atomic data service extraction approach.

Section 4 presents the data service encapsulation framework. Section 5 describes the data service generation system, then evaluates key algorithms and generated data services in detail. Finally, Sect. 6 concludes the paper.

2 Related Work

In 2008, the BEA proposed encapsulating the data into the data service [4], which is similar to the general web service. The data service can be accessed by the interface and output a desired dataset, is one of the hot spots in the current service computing field.

The data service can not only directly access the data source but can also integrate into the SOA through a standard interface [3], which makes up shortcomings of traditional SOA in the data access and provides a new effective approach to integrate multiple datasets on the web [12]. Data services can be extracted from heterogeneous data for easy data management and rapid data manage and sharing in the enterprise systems [17]. The data service also offers an important way to implement the delivery and usage model for various Cloud-based resources and abilities. Wang [16] discusses the issue of streaming data integration and services based on Cloud computing, then summarizes the challenges and future trends. Zhang [19] proposes a method for encapsulating stream sensor data which processes sensor stream data with service modeling operations and distributes data on-demand based on Pub/Sub mechanism.

Besides limitation for the data sharing, traditional approaches in data technologies do not achieve to fully separate software services from data and thus impose limitations in inter-operability. Terzo [14] takes advantage of data service to proposes an approach for sharing and processing of large data collections which abstracts the data location to fully decouple the data and its processing. Liu [10] provides uniform semantic for heterogeneous datasets that different data source database schema mapping a unified global ontology, and then composites data services through the model mapping for solving the interoperability of heterogeneous data. Hong [9] shows a cloud data service architecture for sharing and exchanging multiple data between the server and client in a distributed system which adopts three levels of data security to protect sensitive data. Existing data service platforms, like AquaLogic [2], support data service modeling and help developers to do data queries, which require the users to have certain expertise and relevant rules before they can operate the data services. To make better use of the data service, Vu [15] proposes a description model for data service (DEMODS) which covers all the basic service information for automatic service lookup. Zorrilla [21] describes a data mining service addressed to non-expert data miners which can be delivered as SaaS. Zhang [18] designs a data service API for data analysts to retrieve data that the REST properties and its related hypermedia-driven features are used to generate resource APIs and navigate each other automatically based on analytical needs.

The general trend has been toward low-code and no-code tools that do not require specialized knowledge for data service generation. However, some approaches fall short of the automaticity and increase the learning cost for those who lack of development experience [17]. Moreover, the data service APIs generation driven by user's specific requirements tends to produce tight coupling, which will impact the reusability and scalability of data service [18]. In order to automatically obtain data service from cross-origin datasets, we present a feasible data service generation approach with the aim of supporting multiple application systems for data sharing. Our approach does not require data providers to do additional programming work, and allow data end-users to directly access the data services without any additional constraints.

3 Atomic Data Service Extraction

3.1 Attribute Dependency Graph

Most datasets include a small section known as metadata that contains the contents for understanding the data and its profiling information. The data owner, who wants to publish data services, needs to provide the necessary data connection information. With existing data connectivity APIs, we can obtain the tables, attributes and dependencies from the metadata, which are the basis for data service extraction. For example, Java DatabaseMetaData interfaces[2] provide methods to get metadata of the databases.

Table 1 gives the metadata of two actual datasets which are extracted from an elevator design dataset and an elevator maintenance dataset, and shows their tables and corresponding attributes. Generally, an attribute is the abstract characteristic description of an object, and data are the specific values of an attribute. The data dependencies are the inherent constraint among the attributes.

Table 1. Datasets extracted from two elevator enterprises

Dataset	Table name		Attributes
Design dataset	elevator_info		a; b; c; d;
	client_info		e; f; g; h;
	order_info		i; a′; e′; j;
Maintenance dataset	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

Definition 1 (Functional dependency). *Given a relation R, there exists a functional dependency between two set of attributes X and Y, if, and only if, the X value precisely determines the Y value. It is represented as* $X \rightarrow Y$.

[2] docs.oracle.com/javase/9/docs/api/java/sql/DatabaseMetaData.html.

According to the definition of functional dependency, we can derive the full dependency, partial dependency and inter dependency.

Inference 1: *In a relation, there exists a Full Dependency between two set of attributes X and Y, when Y is dependent on X and is not dependent on any proper subset of X. It is represented as* $X \xrightarrow{f} Y$. *Otherwise, it is a Partial Dependency between X and Y, represented as* $X \xrightarrow{p} Y$.

Inference 2: *In a relation, there exists an Inter Dependency between two set of attributes X and Y, when X and Y are dependent one another. It is represented as* $X \leftrightarrow Y$.

Definition 2 (Join dependency). *A set of attributes X is the common attributes of relation* $R_1(U_1)$ *and relation* $R_2(U_2)$. *If* $X \rightarrow U_2$, *then* U_2 *is join dependent on X.*

We consider the join dependency as a special functional dependency. Then, the functional dependency defines all the inner and outer data dependencies between relations. A dependency graph can be established through functional dependency among attributes, called attribute dependency graph.

Definition 3 (Attribute dependency graph, ADG). *The attribute dependency graph is an extended directed graph that describes the dependencies among attributes. It can be defined as a tuple.*

ADG = (U, E), where $U = \{a_1, a_2, \ldots, a_n\}$, *in which* a_i *is an attribute;* $E = \{e_1, e_2, \ldots, e_m\}$, *in which* $e_i = X \rightarrow a_j$ *represents the* a_j *is dependent on the X* $(X \subseteq U)$.

Figure 1 shows the ADG constructed according to the functional dependency of attributes in Table 1. Each node of the graph represents an attribute, and each arrow of the graph represents a dependency between two nodes. For example, the attributes ***b, c,*** and ***d*** are dependent on the attribute ***a***. The attribute ***a*** is inter-dependent on the attribute ***a'*** and the attributes ***o'*** and ***k'*** are partial dependent on the attribute ***t***. The attributes ***i*** and ***k*** represent the elevator registration number. The semantic equivalence of the two attributes ***i*** and ***k*** provide a bridge for data integration and data sharing.

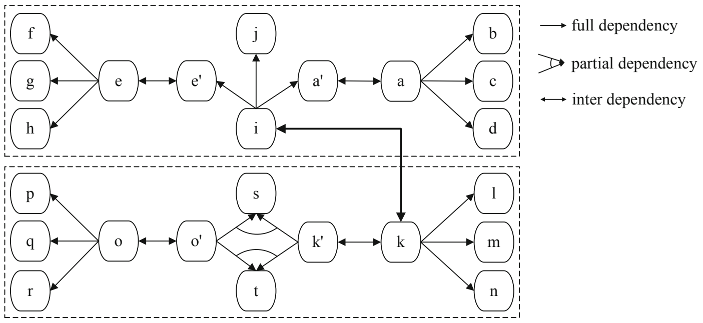


Fig. 1. Attribute dependency graph.

3.2 Atomic Data Service Extraction Algorithm

The data services are extracted from the ADG by encapsulating a set of attributes. The service granularity directly affects the reusability, flexibility and efficiency. If the encapsulated attributes cannot be further subdivided, the corresponding data service is an atomic data service (ADS). We give the formal definition as follows.

Definition 4 (Atomic data service, ADS). *A semantically non-dividable data service is called an atomic data service. Formally, it is a tuple.*

ADS = (ID, Name, Fields, Description, Inputs, Outputs, Operations, Publisher), where ID represents the identification of the ADS; Name represents the name of the ADS; Fields represent the encapsulated attributes of the ADS; Description represents the semantic information of the ADS; Inputs show the multiple input parameters of the ADS; Outputs show the execution result of the ADS; Operations give the possible operations to the ADS; Publisher shows the source of the ADS.

The atomic data service extraction algorithm is shown in Algorithm 1. The input is the metadata of dataset, and the output is the set of ADSs. The attribute set ($U = \{a_1, a_2, ..., a_n\}$) and dependency set ($E = \{e_1, e_2, ..., e_m\}$) are obtained from the metadata. Based on the ADG, the algorithm selects each dependency ($e_i = X \rightarrow a_j$, $X \subseteq U$) in turn and extracts the ADSs with the following rules: (1) Each attribute in X will be extracted as an ADS, whose input and output are the specified attribute; (2) All the attributes in X and the attribute $\boldsymbol{a_j}$ will be extracted as an ADS, whose input is one attribute of X or the attribute $\boldsymbol{a_j}$, and outputs are all the attributes in X and $\boldsymbol{a_j}$.

Algorithm 1. Data service extraction algorithm

```
Input: Metadata of dataset
Output: ADSs
 1: U: The attribute set {$a_1, a_2, …, a_n$}, $a_i$ is an attribute
 2: E: The dependency set {$e_1, e_2, …, e_m$}, $e_i = X \rightarrow a_j$ ($X \subseteq U$)
 3: for Each table in Metadata do
 4:     Add the attributes of table into U
 5:     Add the dependencies into E
 6:     X ← the primary key of table
 7:     for Each attribute in X do
 8:         $ADS_1$ ← attribute
 9:         Add $ADS_1$ into ADSs
10:     end for
11: end for
12: for Each e in E do
13:     $ADS_2$ ← e.X + e.a
14:     Add $ADS_2$ into ADSs
15: end for
16: return ADSs
```

Table 2 lists the ADSs extracted from the ADG of Fig. 1. For example, given the dependency $\{\boldsymbol{k'}, \boldsymbol{o'}\} \rightarrow \boldsymbol{s}$, the attribute $\boldsymbol{k'}$ and $\boldsymbol{o'}$ are extracted as two ADSs individually, i.e. the ADS_{21} and ADS_{22}. In addition, the attributes $\{\boldsymbol{k'}, \boldsymbol{o'}, \boldsymbol{s}\}$ are extracted as one ADS, whose input can be any attribute of $\{\boldsymbol{k'}, \boldsymbol{o'}, \boldsymbol{s}\}$ and outputs are the attributes of $\{\boldsymbol{k'}, \boldsymbol{o'}, \boldsymbol{s}\}$, i.e. the ADS_{23}. According to the above extraction rules, the total extracted ADS number is equal to the total attribute number. The ADS_9 and ADS_{13} encapsulate the attribute $\boldsymbol{i}$ and $\boldsymbol{k}$ respectively which express the same semantics and mainly used for connection. The ADS_2 encapsulates the attributes $\boldsymbol{a}$ and $\boldsymbol{b}$, and used to query the elevator number or the elevator model.

Table 2. Atomic data services extracted from the attribute dependency graph.

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.
…	…	…	…	…	…	…	…
09	ADS_9	{i}	Query registration no	Registration no	Registration no	Get	Design Depart.
…	…	…	…	…	…	…	…
13	ADS_{13}	{k}	Query registration no	Registration no	Registration no	Get	Maintenance Depart.
…	…	…	…	…	…	…	…
21	ADS_{21}	{k′}	Query registration no	Registration no	Registration no	Get	Maintenance Depart.
22	ADS_{22}	{o′}	Query maintenance id	Maintenance id	Maintenance id	Get	Maintenance Depart.
23	ADS_{23}	{k′, o′, s}	Query registration no, maintenance id, maintenance parts	Registration no, maintenance id **or** maintenance parts	Registration no, maintenance id **and** maintenance parts	Get	Maintenance Depart.
24	ADS_{24}	{k′, o′, t}	Query registration no, maintenance id, maintenance price	Registration no, maintenance id **or** maintenance price	Registration no, maintenance id **and** maintenance price	Get	Maintenance Depart.

Since the ADSs are obtained by encapsulating attributes, the inherent data dependencies among attributes can be directly mapped onto the dependencies among data services. We define the data service dependency graph as follows.

Definition 5 (Data service dependency graph, DSDG). *The data service dependency graph is an extended directed graph that describes the dependencies between atomic data services. It can be defined as a tuple.*

DSDG = (DS, E), where DS = {ADS_1, ADS_2, ..., ADS_n}, in which ADS_i is an atomic data service; E = {e_1, e_2, ..., e_m}, in which $e_i = A \rightarrow ADS_j$ represents that ADS_j is dependent on the A ($A \subseteq DS$).

The DSDG of the data services in Table 2 is given in Fig. 2. In essence, the DSDG shows the logical structure of the ADSs and provides a foundation for further data service composition. Based on the service dependency graph, data services can be composed into composite data services to satisfy users' complex data requirements in the future.

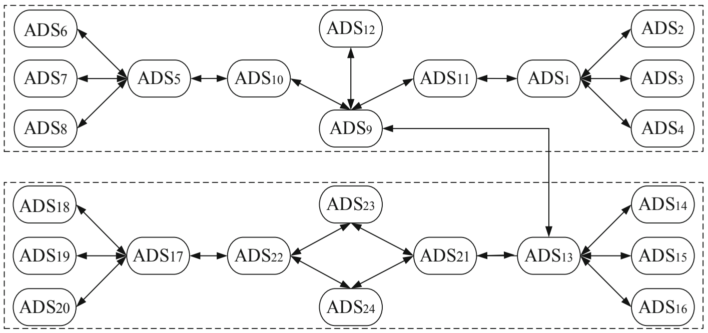


Fig. 2. Data service dependency graph.

4 Data Service Encapsulation

4.1 Data Service Encapsulation Framework

We encapsulate the extracted ADSs into RESTful services, which is based on the representational state transfer (REST) technology. This kind of service defines a set of constraints and properties, and uses HTTP requests to GET, PUT, POST and DELETE resources [13].

We take advantage of the Spring Boot[3] framework to build the RESTful services. Figure 3 shows the data service encapsulation framework. It includes three layers. The entity layer is a mapping of the dataset. An entity object provides a representation of data from a table or other types of data sources. The data access object (DAO) layer provides specific data access operations to the dataset or other persistence mechanism.

[3] http://spring.io/projects/spring-boot/.

The service layer defines the RESTful services that uses the GET operation to retrieve the resource, PUT operation to change or update the resource, the POST operation to create the resource, and the DELETE operation to remove the resource.

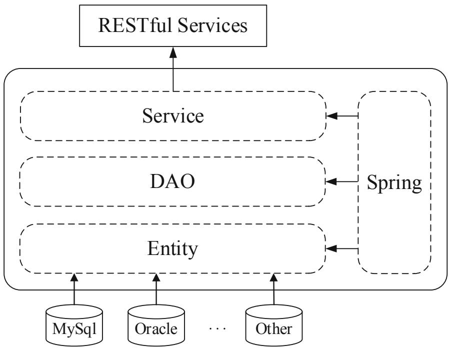


Fig. 3. Data service encapsulation framework.

Based on this framework, we design a reusable template that contains the desired code, placeholders like ${variableName}, and logics like conditionals, loops, etc. Table 3 shows the necessary template files for data service encapsulation.

Table 3. Template files for data service encapsulation.

ID	Template name	Function
1	Entity template	Entity object
2	DAO template	Data access interface and implementation
3	Service template	Data service interface and implementation

The entity template contains the entity object template file. Its fields are all attributes of tables or other data sources, and its methods are the field access methods, such as GET and SET methods. The data type of attributes is unified into String.

The DAO template contains data access interface and implementation details. This template provides abstract interfaces of data operations and the implementation of the interfaces.

The service template contains data service interface and implementation details. The data service interface defines the name, inputs, outputs, operations and URI of the RESTful services. We create RESTful services with CXF[4] that implements the JAX-RS specification and can be easily integrated with the Spring framework.

Figure 4 shows the data service encapsulation procedure. We first extract the meta-data according to the requirements of data service template and generate the data-model

[4] http://cxf.apache.org/.

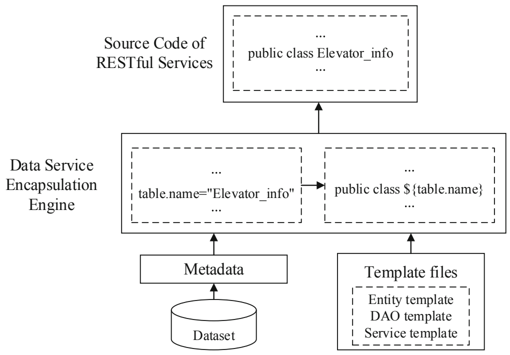

Source Code of RESTful Services
...
public class Elevator_info
...
Data Service Encapsulation Engine
...
table.name="Elevator_info"
...
...
public class ${table.name}
...
Metadata
Dataset
Template files
Entity template
DAO template
Service template

Fig. 4. Data service encapsulation procedure.

that represents the totality of data for template. Then, we utilize a mature template engine, named FreeMarker[5], to parse the data service template files. The engine takes the data-model and data service template files as inputs, then generates the source code of RESTful services by inserting desired code in the template files according to the data-model.

4.2 Data Service Encapsulation Algorithm

The data service encapsulation algorithm is given in Algorithm 2. The input is the metadata of dataset and data service template files, and the output is the source code of RESTful services. The metadata contains all the profiling information of the dataset, such as the tables and the attributes. The table is packed into the data-model of entity template. Combined with the dependencies among attributes, the data-model of DAO template is obtained. According the extracted ADSs, the data-model of service template is organized. Then FreeMarker supplies the data-model for template files in turn and generates the source code of RESTful services. Finally, the source code is compiled to complete the encapsulation.

Taking ADS_2 as an example. This data service has two methods of the GET operation. One is the *getElevatorInfoElevatorModel*. Its input is the elevator no, and its outputs are the list of elevator no and elevator model. The other is the *getElevatorInfoByElevatorModel*. Its input is the elevator model, and its outputs are the list of elevator no and elevator model. The interface source code of the ADS_2 is shown as follow.

5 https://freemarker.apache.org/.

Algorithm 2. Data service encapsulation algorithm

```
Input: Metadata of dataset, data service template files
Output: Source code of the RESTful services
 1: Objects: Entity objects from Metadata
 2: Entity template: entity object template file
 3: DAO template: data access interface and implementation template files
 4: Service template: data service interface and implementation template files
 5: for Each object in Objects do
 6:     Get all attributes and the dependencies among attributes of object
 7:     entity_data ← get the data-model of Entity template
 8:     Source code ← Entity template + entity_data
 9:     dao_data ← get the data-model of DAO template
10:     Source code ← DAO template + dao_data
11:     service_data ← get the data-model of Service template
12:     Source code ← Service template + service_data
13: end for
14: return Source code
```

```
@Path("/ElevatorInfo") @Produces({MediaType.APPLICATION_JSON+"; charset=UTF-8"})
public interface ElevatorInfoService {
    @GET @Path("/ElevatorModel/{elevatorNo}")
    List<ElevatorInfo> getElevatorInfoElevatorModel (@PathParam(value="elevatorNo") String ele-
vatorNo);
    @GET @Path("/ElevatorNo/ElevatorModel/{elevatorModel}")
    List<ElevatorInfo> getElevatorInfoByElevatorModel (@PathParam(value="elevatorModel")
String elevatorModel);
    …}
```

Corresponding to the interface of the ADS_2, the implementation source code of the ADS_2 is shown as follow.

```
@Component ("elevatorInfoService")
public class ElevatorInfoServiceImpl implements ElevatorInfoService {
   @Resource
   private ElevatorInfoDao elevatorInfoDao;
   public List<ElevatorInfo> getElevatorInfoElevatorModel (String elevatorNo) {
        return elevatorInfoDao.getElevatorInfoElevatorModel (elevatorNo);}
   public List<ElevatorInfo> getElevatorInfoByElevatorModel (String elevatorModel) {
        return elevatorInfoDao.getElevatorInfoByElevatorModel (elevatorModel);}
   …}
```

5 Experimental Results

5.1 Prototype System Development

We have developed a data service generation system. Currently, the main functions of the system include dataset management, data service extraction, data service encapsulation, and data service management. Cross-origin datasets from the elevator enterprises which are medium-sized and poor sharing ability of data resources, have been utilized as test data in current system. The datasets include a design dataset, a sales dataset, a customer dataset, a manufacturing dataset, and a maintenance dataset.

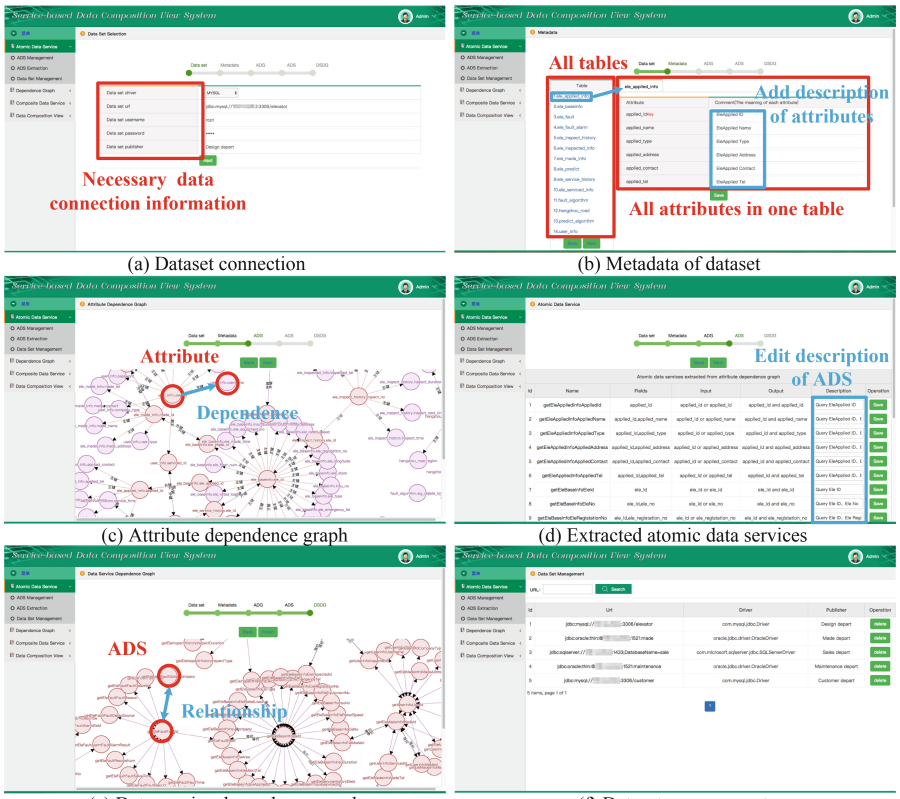
Take the elevator design dataset stored in MySQL as an example, Fig. 5 shows the main interfaces of data service extraction. Figure 5(a) shows the interface of obtaining the necessary data connection information provided by the data owner, such as the URL (jdbc:mysql://<host>:<port>/<database_name>), the data driver (com.mysql.jdbc.Driver), username and password, to access a specific dataset. With the methods in Java DatabaseMetaData interfaces, like getTables(), getColumns() and getPrimaryKeys(), the system can obtain all tables, attributes and dependencies among attributes. The description of attributes can be further added to make end-user better understand in Fig. 5(b). Then, the attribute dependency graph (ADG) shown in Fig. 5(c) is built. According to the Algorithm 1, the system will extract the atomic data services (ADSs) based on the ADG. The extracted ADSs are listed in Fig. 5(d). The default description of ADSs is encapsulated attributes and can be edited by end-user. Figure 5(e) shows the relationships among ADSs, where a data service dependency graph (DSDG) is built. After that, the data service extraction is completed. The extracted datasets are shown in the Fig. 5(f).

Figure 6 shows the main interfaces of data service encapsulation. According to Algorithm 2, the system will encapsulate the extracted ADSs into RESTful services using the encapsulation framework. Then, the data services can be deployed on any server node specified by the data owner to ensure service availability, as shown in Fig. 6(a). The data owner can temporarily shutdown or permanently remove the ADSs which involve sensitive date for data security. End-user can access the data services with the provided URL, and get the desired dataset, as shown in Fig. 6(b). These encapsulated ADSs can be directly accessed, and can also be further composed into composite data services to acquire more complex data.

5.2 System Evaluation

In this subsection, we evaluate the two key algorithms adopted in the system: the data service extraction algorithm, and the data service encapsulation algorithm. The experimental hardware is a 2.50 GHz 8-core CPU, 16 G 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.

We select five different experimental datasets [20] for the evaluation. Table 4 shows the information of the datasets including the total number of tables and attributes, the total number of ADSs extracted by Algorithm 1 and total number of source files generated by Algorithm 2.



Data Set Selection

Data set driver	MYSQL
Data set url	jdbc:mysql://[redacted]2:3306/elevator
Data set username	root
Data set password	••••
Data set publisher	Design depart

(a) Dataset connection

Metadata

Table
1.ele_applied_info
2.ele_baseinfo
3.ele_fault
4.ele_fault_alarm
5.ele_inspect_history
6.ele_inspected_info
7.ele_made_info
8.ele_predict
9.ele_service_history
10.ele_serviced_info
11.fault_algorithm
12.hangzhou_road
13.predict_algorithm
14.user_info

ele_applied_info

Attribute	Comment(The meaning of each attribute)
applied_idKey	EleApplied ID
applied_name	EleApplied Name
applied_type	EleApplied Type
applied_address	EleApplied Address
applied_contact	EleApplied Contact
applied_tel	EleApplied Tel

(b) Metadata of dataset

Attribute Dependence Graph

(c) Attribute dependence graph

Atomic Data Service

Atomic data services extracted from attribute dependence graph

Id	Name	Fields	Input	Output	Description	Operation
1	getEleAppliedInfoAppliedId	applied_id	applied_id or applied_id	applied_id and applied_id	Query EleApplied ID	Save
2	getEleAppliedInfoAppliedName	applied_id,applied_name	applied_id or applied_name	applied_id and applied_name	Query EleApplied ID、E	Save
3	getEleAppliedInfoAppliedType	applied_id,applied_type	applied_id or applied_type	applied_id and applied_type	Query EleApplied ID、E	Save
4	getEleAppliedInfoAppliedAddress	applied_id,applied_address	applied_id or applied_address	applied_id and applied_address	Query EleApplied ID、E	Save
5	getEleAppliedInfoAppliedContact	applied_id,applied_contact	applied_id or applied_contact	applied_id and applied_contact	Query EleApplied ID、E	Save
6	getEleAppliedInfoAppliedTel	applied_id,applied_tel	applied_id or applied_tel	applied_id and applied_tel	Query EleApplied ID、E	Save
7	getEleBaseinfoEleId	ele_id	ele_id or ele_id	ele_id and ele_id	Query Ele ID	Save
8	getEleBaseInfoEleNo	ele_id,ele_no	ele_id or ele_no	ele_id and ele_no	Query Ele ID、Ele No	Save
9	getEleBaseInfoEleRegistationNo	ele_id,ele_registation_no	ele_id or ele_registation_no	ele_id and ele_registation_no	Query Ele ID、Ele Regi	Save

(d) Extracted atomic data services

Data Service Dependence Graph

(e) Data service dependence graph

Data Set Management

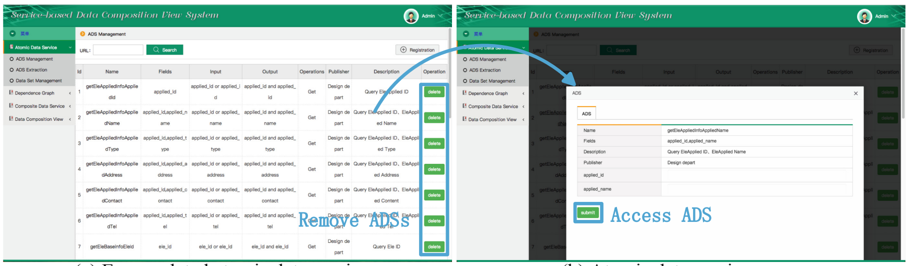
URL: Search

Id	Url	Driver	Publisher	Operation
1	jdbc:mysql://[redacted]:3306/elevator	com.mysql.jdbc.Driver	Design depart	delete
2	jdbc:oracle:thin:@[redacted]:1521:made	oracle.jdbc.driver.OracleDriver	Made depart	delete
3	jdbc:sqlserver://[redacted]:1433;DatabaseName=sale	com.microsoft.sqlserver.jdbc.SQLServerDriver	Sales depart	delete
4	jdbc:oracle:thin:@[redacted]:1521:maintenance	oracle.jdbc.driver.OracleDriver	Maintenance depart	delete
5	jdbc:mysql://[redacted]:3306/customer	com.mysql.jdbc.Driver	Customer depart	delete

5 items, page 1 of 1

(f) Dataset management

Fig. 5. Main interfaces of data service extraction.



ADS Management

URL: Search

Id	Name	Fields	Input	Output	Operations	Publisher	Description	Operation
1	getEleAppliedInfoAppliedId	applied_id	applied_id or applied_id	applied_id and applied_id	Get	Design depart	Query EleApplied ID	delete
2	getEleAppliedInfoAppliedName	applied_id,applied_name	applied_id or applied_name	applied_id and applied_name	Get	Design depart	Query EleApplied ID、EleApplied Name	delete
3	getEleAppliedInfoAppliedType	applied_id,applied_type	applied_id or applied_type	applied_id and applied_type	Get	Design depart	Query EleApplied ID、EleApplied Type	delete
4	getEleAppliedInfoAppliedAddress	applied_id,applied_address	applied_id or applied_address	applied_id and applied_address	Get	Design depart	Query EleApplied ID、EleApplied Address	delete
5	getEleAppliedInfoAppliedContact	applied_id,applied_contact	applied_id or applied_contact	applied_id and applied_contact	Get	Design depart	Query EleApplied ID、EleApplied Content	delete
6	getEleAppliedInfoAppliedTel	applied_id,applied_tel	applied_id or applied_tel	applied_id and applied_tel	Get	Design depart	Query EleApplied ID、EleApplied Tel	delete
7	getEleBaseinfoEleId	ele_id	ele_id or ele_id	ele_id and ele_id	Get	Design depart	Query Ele ID	delete

(a) Encapsulated atomic data services

ADS

Name	getEleAppliedInfoAppliedName
Fields	applied_id,applied_name
Description	Query EleApplied ID、EleApplied Name
Publisher	Design depart
applied_id	
applied_name	

(b) Atomic data service access

Fig. 6. Main interfaces of data service encapsulation.

We first evaluate the system performance of the metadata extraction. Figure 7 shows the overall time consumed to complete the extraction by varying the number of tables and attributes, where the X axis represents the total number of tables, Y axis represents the total number of attributes and the Z axis represents the execution time.

Table 4. Experimental datasets for data service extraction and encapsulation algorithm.

Datasets	Total table number	Total field number	Total ADS number	Total source file number
Customer dataset	14	120	120	74
Sales dataset	27	231	231	139
Made dataset	37	317	317	189
Maintenance dataset	47	400	400	239
Design dataset	62	473	473	314

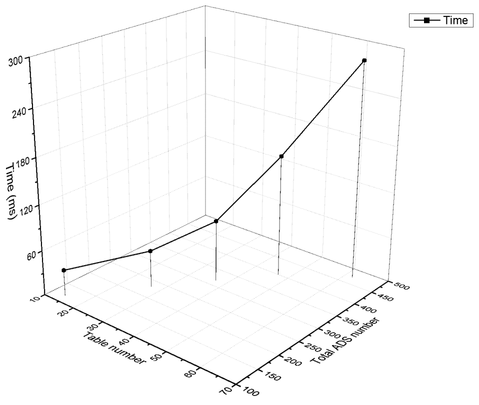


Fig. 7. Performance of the metadata extraction.

It shows that metadata can be obtained in a short time and more attributes in the tables require more time to deal with dependencies among attributes.

Then, we evaluate the system performance that aims to show the overall time consumed by the ADS extraction and encapsulation algorithm. Figure 8(a) shows the overall time consumed to complete the extraction by varying the ADS number, where the X axis represents the total number of ADSs and the Y axis represents the execution time. It can be seen that the execution time rises slightly with the increasing number of the ADSs. Figure 8(b) shows the overall time required to complete the service encapsulation with different number of source files, where the X axis represents the total number of source files and the Y axis represents the execution time. The encapsulation algorithm consumes obviously more time than the extraction algorithm. The reason is that the service encapsulation involves I/O time due to file reading and writing. Totally, the system has high efficiency for both service extraction and service encapsulation.

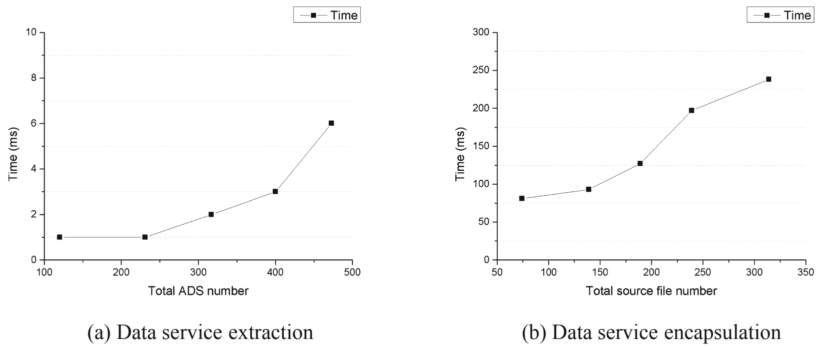


Fig. 8. Performance of the data service extraction and encapsulation algorithm.

5.3 Analysis

Maturity. We use a model of restful maturity that was developed by Leonard Richardson, called Richardson Maturity Model [11], to evaluate the RESTfulness of our data service APIs. The RESTful services are classified into four levels of maturity according to the design constraints of the REST architectural style: (1) Level 0: Tunneling messages through an open HTTP port leads only to the basic ability to communicate and exchange data. (2) Level 1: Making use of multiple identifiers to distinguish different resources. (3) Level 2: Making proper use of the REST uniform interface in general and of the HTTP verbs in particular. (4) Level 3: In addition to exposing multiple addressable resources which share the same uniform interface also make use of hypermedia to model relationship between resources. Compared with the service APIs of Microsoft's OData (Open Data Protocol)[6] which are not up to the highest level of REST for the reason that there is no navigation for services to include links or self-documentation in response [18], our approach has the capacity to return a response body with a set of resources which associated with the user's request based on the data service dependency graph model. Therefore, our data services reach the maturity level 3.

Discoverability. Discoverability means that the desired data services can be accessed when user requests a resource, which facilitates the composition of data services. Compared with the conventional service interfaces that the process of service discovery is fallible and time-consuming, such as Twitter APIs, our approach can automatically obtain the user-desired data services by searching the data service dependency graph which enables achieving the discoverability.

Reusability. Reusability means that the data services can be reused multiple times without configuration or minor changes. Compared with [18] that there is a one-to-one correspondence between data services and user's data needs, our approach provides a finer granularity and a higher degree of flexibility, which can be composed according to

6 https://www.odata.org/.

various kinds of requirements without replaceability of bindings or binding configuration. The flexible and extensible data services allow users to customize the composite data services for specific requirements which benefits to improve the reusability.

6 Conclusion

To automatically obtain data services from cross-origin datasets, we presented an automatic data service generation approach. The attribute dependency graph (ADG) was built according to inherent data dependencies among dataset. Based on the ADG, data services could be automatically extracted. Then, we designed a data service encapsulation framework. It could automatically encapsulate the extracted ADSs into RESTful services with a specific implementation template. We have developed a data service generation system, and actual cross-origin datasets have been carried out in the system to generate data services. We also evaluated the performance of the extraction and encapsulation algorithms, then demonstrated our data services in maturity, discoverability and reusability. In the future, we will focus on the research of automatic data service generation for unstructured data to improve the availability.

Acknowledgments. The authors gratefully acknowledge the support from the Zhejiang Natural Science Foundation, China (No. LY19F020034).

References

1. Abiteboul, S., Benjelloun, O., Milo, T.: Web services and data integration. In: International Conference on Web Information Systems Engineering, pp. 3–6 (2002)
2. Borkar, V., et al.: Graphical XQuery in the aqualogic data services platform. In: ACM SIGMOD International Conference on Management of Data, pp. 1069–1080 (2010)
3. Carey, M.: Declarative data services: this is your data on SOA. In: IEEE International Conference on Service-Oriented Computing and Applications, p. 4 (2007)
4. Carey, M., Reveliotis, P., Thatte, S., Westmann, T.: Data service modeling in the aqualogic data services platform. In: Services, pp. 78–80 (2008)
5. Carey, M.J., Onose, N., Petropoulos, M.: Data services. Commun. ACM **55**(6), 86–97 (2012)
6. Chen, C.L.P., Zhang, C.Y.: Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf. Sci. **275**(11), 314–347 (2014)
7. Dillon, S., Stahl, F., Vossen, G.: Towards the web in your pocket: curated data as a service. In: International Conference on Computational Collective Intelligence Technologies and Applications, pp. 25–34 (2012)
8. Halevy, A.Y., Rajaraman, A., Ordille, J.J.: Data integration: the teenage years. In: International Conference on Very Large Data Bases (2006)
9. Hong, X., Rong, C.M.: Multiple data integration service. In: International Conference on Advanced Information NETWORKING and Applications Workshops, pp. 860–865 (2014)
10. Liu, X., Hu, C., Li, Y., Jia, L.: The advanced data service architecture for modern enterprise information system. In: International Conference on Information Science and Applications, pp. 1–4 (2014)

11. Pautasso, C.: RESTful web services: principles, patterns, emerging technologies. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) Web Services Foundations, pp. 31–51. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-7518-7_2
12. Rajesh, S., Swapna, S., Reddy, P.: Data as a service (DaaS) in cloud computing. Glob. J. Comput. Sci. Technol. **12**(11), 25–29 (2012)
13. Richardson, L., Ruby, S.: Restful Web Services, pp. 199–204. O'Reilly Media Inc., Sebastopol (2007)
14. Terzo, O., Ruiu, P., Bucci, E., Xhafa, F.: Data as a service (DaaS) for sharing and processing of large data collections in the cloud. In: Seventh International Conference on Complex, Intelligent, and Software Intensive Systems, pp. 475–480 (2013)
15. Vu, Q.H., Pham, T.V., Truong, H.L., Dustdar, S., Asal, R.: Demods: a description model for data-as-a-service. In: IEEE International Conference on Advanced Information Networking and Applications, pp. 605–612 (2012)
16. Wang, G.L., Han, Y.B., Zhang, Z.M., Zhu, M.L.: Cloud-based integration and service of streaming data. Chin. J. Comput. 107–125 (2017)
17. Yu, H., Cai, H., Zhou, J., Jiang, L.: Data service generation framework from heterogeneous printed forms using semantic link discovery. Future Gener. Comput. Syst. **79**, 514–527 (2017)
18. Zhang, Y., Zhu, L., Xu, X., Chen, S., Tran, A.B.: Data service API design for data analytics. In: Ferreira, J.E., Spanoudakis, G., Ma, Y., Zhang, L.-J. (eds.) SCC 2018. LNCS, vol. 10969, pp. 87–102. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94376-3_6
19. Zhang, Z.M., Liu, C., Su, S., Zhang, S.L., Han, Y.B.: SDaaS: a method for encapsulating sensor stream data as services. Chin. J. Comput. **40**(2), 445–463 (2017)
20. Zhang, Z.J., Zhang, Y.M., Lu, J.W., Xu, X.S., Gao, F., Xiao, G.: CMfgIA: a cloud manufacturing application mode for industry alliance. Int. J. Adv. Manuf. Technol. **98**(9–12), 2967–2985 (2018)
21. Zorrilla, M., Garca-Saiz, D.: A service oriented architecture to provide data mining services for non-expert data miners. Decis. Support Syst. **55**(1), 399–411 (2013)