Data Service API Design for Data Analytics

Yun Zhang[1,2(✉)], Liming Zhu[1,2], Xiwei Xu[1,2], Shiping Chen[1,2], and An Binh Tran[1]

[1] School of Computer Science and Engineering, UNSW, Sydney, Australia
[2] Data 61, CSIRO, Sydney, Australia
{yun.zhang,liming.zhu,xiwei.xu,shiping.chen}@data61.csiro.au

Abstract. Data service APIs provide uniform and filtered interfaces for data analysts to retrieve data. However, existing RESTful data services do not serve data analytics well because most of them are designed based on the underlying data schema rather than aligning with the requirements of data analytics. First, the API representations only support *one-off* communication, which lacks analytic semantics to guide analysts to continuously explore and retrieve data. Second, the current data service design does not support re-usage of data exploration processes and derived data generated from data analysts.

In this paper, we propose an analytics-focused API design for data services. First, we introduce a service architecture and its resource APIs to realize core functions of data retrieval. Second, we design a navigation model for analysts to navigate resource APIs more efficiently. Third, we extend and leverage data package technique to provide context information about the origin, scope, and historical manipulations on a certain dataset. This mechanism allows the analysts to share and reuse historical data exploration process and derived data. We evaluate our approach using a case study and compare our approach against the conventional data APIs. The evaluation shows that our approach has advantages over traditional data service APIs in maturity, interoperability, discoverability, and reusability.

Keywords: Data analytics · Data service · REST · API
Data package

1 Introduction

Large amounts of data are increasingly being published on the web. For example, on Twitter, more than three million tweets are published every 10 min[1]. Another example is open data platforms provided by governments, like data.gov.au[2], which already published over twenty thousands of datasets for free. How to

[1] http://www.internetlivestats.com/twitter-statistics/.
[2] https://data.gov.au.

J. E. Ferreira et al. (Eds.): SCC 2018, LNCS 10969, pp. 87–102, 2018.
https://doi.org/10.1007/978-3-319-94376-3_6

properly and efficiently retrieve these data for data analytics is becoming a hot issue [1].

To analyze a dataset, data analysts often start by exploring data before knowing exactly what they are looking for. It is not pragmatic to download the whole large dataset before performing any exploratory analysis. Instead, it is desirable to allow analysts to have a glimpse into the data through a sequence of exploratory queries before the retrieval for further analysis [2]. Data exploration, which is an interactive process to retrieve data, allows the analysts issue a query, receive a response, and then iteratively interact with the data system to refine their query based on the response from the system and domain knowledge [3].

Since data exploration is labor-intensive and repetitive, it would be beneficial for analysts if the value-added data derived from the exploration stage could be shared and reused in future. Data analysts can share the results from the earlier exploration to better streamline the data analytics pipeline. To enable more efficient data sharing and reuse, it is very important to provide provenance information of data source so that data consumers are informed about what sort of earlier manipulations have been done to the data.

Data services provide uniform, scalable, and filtered interfaces for data analysts to retrieve data [4]. Many companies and platforms, like Twitter, Google, and CKAN[3] offer data service APIs that provide simple and easy-to-use access to some of their resources. Data services allow third-parties to easily integrate the data resource into their applications. However, these conventional interfaces fall short on supporting responsive, interactive, and comprehensive data retrieval for analytics. First, the existing data services are designed to answer questions according to the underlying database schema and pre-assembled index, rather than being driven by the requirement of data retrieval for data analytics [5]. Second, the current data services only support *one-off* queries, which are isolated, static and of not analytics semantics. Data analyst have to blindly request data services many times to understand underlying data. Third, there is no standard mechanism to provide context information about the origin, scope, and usage of the data in data services. Data analysts cannot be informed about what data exists, how the data is derived and used, and as a result, they cannot infer whether these processed data can be reused.

In this paper, we propose a data service API design driven by the requirements of data analytics. Our contributions include (1) a new data service architecture with a set of analytic APIs (2) a navigation model to help discover and generate data service API dynamically, and (3) a mechanism that extends *data package* to share data processing scripts and data context information associated with data. We evaluate the proposed API design through a real case study and discuss quality attributes. The evaluation shows that our approach has advantages over conventional data service APIs in maturity, interoperability, discoverability, and reusability.

The remainder of this paper is organized as follows. Section 2 describe a brief background and some related work. Section 3 gives three research requirements

[3] https://ckan.org/.

derived from a scenario of human resource data analytics. Section 4 introduces our approach with the service architecture, the navigation model and the application of data package. Section 5 uses a case study to evaluate our API design and discuss the contribution. Section 6 concludes the paper and extended work.

2 Background and Related Work

REST (REpresentational State Transfer) [6] is an architectural style for designing web applications. Following REST design principles, a data service is identified by a URI as a resource. Client applications interact with data services through using request-response messages. Protocols and Structures for Inference (PSI)[4] specification defines a RESTful architecture for presenting concepts used in machine learning as RESTful web services. The data source is wrapped as data service, named relation. However, the relation does not indicate how to discover a related data service for data exploration.

Database-as-a-Service (DaaS) has emerged as a new paradigm in the cloud computing environment. Many commercial databases, like Amazon SimpleDB[5], provide accessible data service APIs to their data stores. HTSQL[6] enables accessing SQLServer via HTTP arbitrarily, which is an advanced query language on the web. However, the design of these CRUD-based data services are merely based on the underlying database schemas and pre-assembled indexes without referring to the domain application protocol.

To build a domain application protocol over HTTP, which is domain agnostic in the web application, additional explicit semantics are needed [7]. In the Semantic Web, semantics are described by ontologies written in RDFS and OWL, while RESTful implementations encode semantics by annotating hypermedia with link relations [8]. Hypermedia as the Engine of Application State (HATEOAS) is a constraint applied to hypermedia. HATEOAS requires that the service embed links in its responses. The links represent the next possible actions that clients can take [6].

However, there is a semantic gap for the clients to be navigated by hyperlink automatically. To remedy this design flaws in HATEOAS implementation, some hypermedia specifications such as AtomPub[7]and OpenSearch[8]are tailored to achieve the specific application goals. However, the semantics of these domain-specific media types are implicit and generic [9]. AtomPub is designed to cover all the collection-based APIs but cannot reflect different application semantics. Microsoft's Open Data Protocol (OData)[9] is derived from AtomPub. OData defines the protocol semantics for filtering and sorting a collection of data, using a query language similar to SQL, but the semantics of the relationship between

[4] http://psi.cecs.anu.edu.au/.
[5] http://aws.amazon.com/simpledb/.
[6] http://htsql.org/.
[7] https://bitworking.org/projects/atom/rfc5023.html.
[8] http://www.opensearch.org/Home.
[9] http://www.odata.org/.

resources focus on data instead of analytics operations, and the related metadata services only present limited description documents [10].

Our approach fills this semantic gap by specifying the domain application protocol for data exploration in analytics and using this protocol for the implementation of our services following HATEOAS principle.

In REST architecture, metadata provides self-describing information about web resources, which enable automatic processing of web resources [11]. However, the metadata is sent in the header of HTTP messages and restricted to provide information about the syntax used in the resource representation. In addition, the semantics of the origin, scope, and usage of the data is less considered. Ground [12] is a data context service that supports collecting, publishing and querying the metadata information from applications, behavior, and change of data context, but it is implemented as a system without consideration of RESTful API presentation. A data package is a collection of datasets, metadata information and other data files. Data package protocol[10] defines an open standard for the format of a package, which guides users to share and manage distributed dataset using data package. However, there is no guidance on how to apply data package for exchanging metadata in RESTful services.

3 Scenario and Requirements

To explore the potential of our data services, we describe a practical human resource analytics scenario in which the data services can aid in data exploration and facilitate better collaboration between data analysts.

The purpose of the analysis is to help a company understand why some of their most experienced employees are leaving prematurely and predict who will leave in future.

Analyst Bob first requests the data service to investigate the sum, average, min, max, and medium of numeric attributes respectively. Then he makes the second request to discover a correlation between each pair of attributes. The results show that on average, employees who left the company have lower satisfaction levels. After knowing all the features of employees who have left, Bob request data service to retrieve the data about valuable but left employees with an evaluation result above average performance, or spend at least four years in the company, or were working on more than five projects at the same time and still left the company. Later, Bob will use these data to conduct an analysis model to predict who will leave. After Bob completes the explorative analysis, this value-added data and his explorative process could be shared by another analyst named Alice. Alice can use Bob's data to do a further analytic activity without preparing the data from scratch. She also can reproduce Bob's exploration process to verify his result. Even, she can extend and construct her operation based on Bob's. Afterward, Alice's data exploration process and derived data can be shared with another analyst.

[10] https://specs.frictionlessdata.io/data-package/.

From this scenario, we can derive three main requirements for building data services to explore and retrieve data.

R1. The data service should be grounded on data explorative operations. An analyst should be able to interact with the data services in the same manner as client side data analysis tools.

R2. The data services should be able to guide analysts to discover the related resources based on their specific requirements. An analyst should be able to navigate resource APIs to understand underlying data efficiently.

R3. The data services should allow data analysts to share and reuse the result of the explorative analysis. An analyst should be able to replay the shared process based on the provided context information about who, when and what operations have been performed on the data.

4 Service Design

The requirements described above motivate us to design a new data service that fills the gap in current data services. First, we introduce a RESTful service architecture including key resources to facilitate data operations in data exploration. Second, we propose a navigation model to describe the relationship of the main resources in the domain of data analytics and illustrate how to use this model to guide analysts to explore data. Finally, we customize and leverage data package into our data services and showcase its usage and advantages for building a data analysis sharing environment.

4.1 Service Architecture

As shown in Fig. 1, the main resources in our architecture are *gateway, filter, aggregator, sampler, function supplier* and *packager*. A request from user or web client is sent to *gateway*, which responds with a roadmap of all resource categories provided by the data service. Each resource category contains a set of RESTful resource APIs that map a category of functions used in data exploration and retrieval. The resources are interconnected via hyperlink whereby the

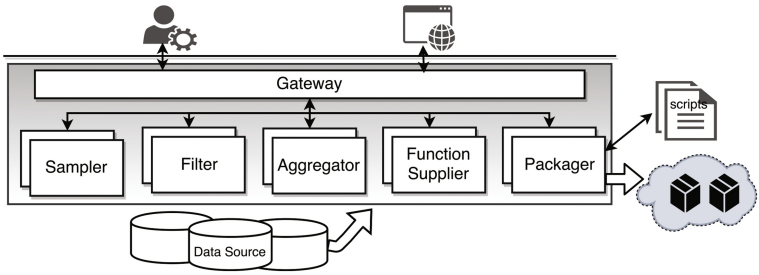

Fig. 1. Data service architecture

user can navigate among resource APIs to explore data. After explorative data analysis done, *packager* created a data package which contains the derived data, processing scripts, and context information. Specifically, the properties of this architecture are in the following:

Gateway is a self-describing resource that exposes the metadata defining data schema and other information like type of data source(e.g. dynamic, static), data size and description about this dataset, which helps the data analyst have an initial understanding about the dataset. Data analysts interact with Gateway to discover the resources related to the dataset.

Filter allows the data analysts to filter the dataset based on dimensions and measures which are presented based on non-numeric attributes and quantitative attributes. Dimensions represent which attributes of data that can be extracted while measures represent the query schema that is used to extract the subset or transformation of the dataset.

Aggregator allows the data analysts to have a summary statistics of the data without extracting original data items. It can perform aggregation function over one attribute's values and group by the attribute's name. The aggregation queries have constraints based on the numeric or non-numeric data attribute.

Sampler provides diverse sampling methods to allow data analysts to quickly build and test their models within a sample of data that can fit into their memory. The sample size and specific sample methods are defined by users.

Function Supplier provides a set of statistics functions, which effectively assist data analysts to discover the relation, general trend, and outlier of the data as auxiliary means. For example, *correlation* is for understanding the relationship between attributes; *isnull* is used to check for the missing value to help analysts estimate the data quality.

Packager retrieves the data, wraps them with optional primitive operations which are presented as scripts into the package, and then stores them in external storage. Using the packager resource, the data consumer can acquire a targeted subset of a dataset in batch along with the optional scripts provided by the data publisher. These scripts can be used to pre-process the data and accelerate the forthcoming analysis work. We will discuss more detail in Sect. 4.3.

4.2 Navigation Model

The resources in the service architecture are interconnected to each other according to the context of the data analytics. A navigation model which defines the domain application protocol for data exploration is designed to assist clients to form their queries for interactive exploring large datasets. Based on the navigation model, data service can recommend next steps for the data analyst in the query session, and provide the information of the relationship between the resources in the context of data analytics.

The navigation model is shown in Fig. 2. The circles represent the resources introduced in the Sect. 4.1, while arrows correspond to the connections between the resource API templates. The relationships across resources are categorized

into four types, including *narrow down*, *summary*, *relate* and *wrap up*. Specifically, *narrow down* means zoom into the data from less detail to more detail. Users could be guided to the *filter* API template by the *narrow down* link to query detailed data from summarized data based on the data distribution or extreme value provided by the *aggregator* or the *sampler*. Conversely, users can zoom out the data that are of little interest to discover other attributes through *sampler* or *aggregator* API template guided by the *summary* link. The *relate* link presents auxiliary services, for example, some statistic functions like correlation, standardization and distribution. During the process of data exploration, *wrap up* appears in every stage to refer users to the *packager* API template when the returned data are too large for the client memory or the users wants to download the whole data with previously recorded data exploration track.

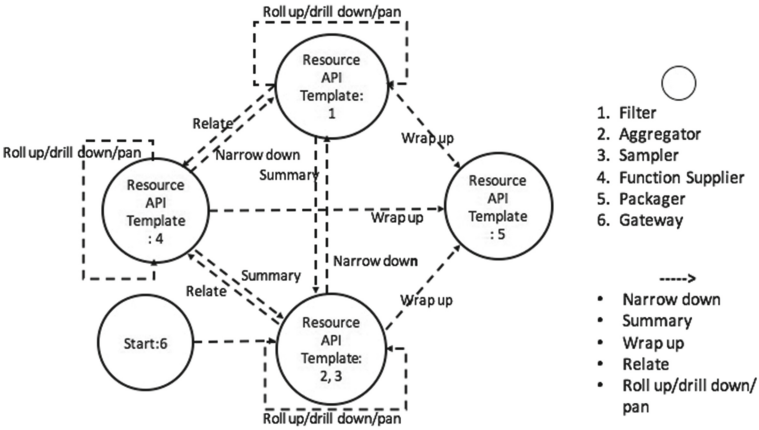

Fig. 2. Navigation model

When focusing on one resource, the user can send a sequence of requests to the resource API template adjusting the parameter values until she is satisfied with the results. Alternatively, such a query session could be accelerated by our navigation model through parameter prediction—parameter values can be used to instantiate the API templates through analyzing the past parameters provided by user. We generalize three types of relationship based on users navigation activates including *Roll up, drill down* and *pan*. Concretely, *drill down* provides a more detailed view by either stepping down a hierarchy within a dimension or introducing additional dimensions through changing the parameters. For example, when viewing the salary data of Australia, a *drill down* link provides the service querying the data of different states like NSW (New South Wales), QLD (Queensland), etc. A further *drill down* on NSW may display data of Sydney. It also can restrict the results in *aggregator* by tweaking the conditions. *Roll up* is the reverse of *drill down*: it means climbing up a concept hierarchy for a dimension, reducing the dimensions or relieving the conditions in a measure.

Pan allows users to change the angle they observed by changing the dimensions of data or the operations used.

The navigation model incorporates the analytics domain semantics into HATEOAS. Specifically, the `links` property is used to represent all the actions and resources related to each resource. Users can select one of the `links` to follow as the next step. `links` is defined as an array of Linked Description Objects (LDOs) in JSON Hyper-Schema[11], which obeys HATEOAS principle and assists discovering all the related resource API templates with the current resource API. Each LDO at least contains one `href` property, which is the target of the link, and a `rel` property indicating the relationship between the linked resource and the current resource. Users could effectively make a data exploration by following `links` embedded in the representation to access the next useful resource.

The navigation model involves both dynamical discovery and generation, which enables users to dynamically discover the resource API templates and automatically generate the parameters of API templates based on the previous input from the client. We present their schemas of Links separately in Listings 1.1 and 1.2.

```
{       "$schema": "http://json-schema.org/draft-04/schema#",
        "title": "Schema defining links between resources",
        "type": "array",
        "items": {
        "links": [{
                "rel": "narrow down",
                "href": "/filter",
                "method": "GET"
        },{
                "rel": "summary",
                "href": "/aggregator",
                "method": "GET"
        },{
                "rel": "relate",
                "href": "/functionSupplier",
                "method": "GET"
        },{
                "rel": "wrap up",
                "href": "/packager",
                "method": "POST",
                "schema": {}
        }]}}
```

Listing 1.1. HyperSchema of links for dynamical discovery

```
{       "$schema": "http://json-schema.org/draft-04/schema#",
        "title": "Schema defining links within one resource",
        "base": "/{resource}?{measures,dimensions}",
        "type": "array",
        "links": [{
```

[11] http://json-schema.org/latest/json-schema-hypermedia.html.

```
        "rel": "drill down",
        "href": "/{resource}?{measures,added_dimensions}",
        "method": "GET"
},{
        "rel": "roll up",
        "href": "/{resource}",
        "method": "GET"
},{
        "rel": "pan",
        "href": "/{resource}?{new_measures,new_dimensions}",
        "method": "GET" }]}...
```

Listing 1.2. HyperSchema of links for dynamical generation

As shown in Listing 1.1, each `links` comprises of `rel` that presents the meaning of related action, and `href` that points to the location of resource. The value of `rel` can be *relate, summary, narrow down* and *wrap up*. `rel` is used for the dynamic discovery of resource APIs. The `method` and `schema` properties specify the HTTP method and data format for the input. Client can send a HTTP OPTIONS request to acquire further assistance on how to form a specific API.

Listing 1.2 defines the schema of `links` for generating the specific resource APIs. According to the different semantics of `rel`, the new parameters can be generated based on the measures and dimensions in the `base`, and form a new resource API as a `href` property for client. The value of `rel` can be *roll up, drill down* and *pan*. `rel` is used for resource APIs dynamic generation.

4.3 Data Package

To allow users to share and reuse the data exploration process, our data services adopt data package as a media type, which has an flexible and extensive data structure to include various data.

Figure 3 gives an overview of the data format of the extended data package, which may contain (1) **data** such as tables and files stored remotely in cloud storage or internally in the package. Data is classified into source data, result data, query data according to their purposes, (2) **scripts** processing and analyzing data, which are written by the data provider or generalized by the data service in any cross-platform languages like Python or Java, and (3) **metadata** describing the structure and the content of the package, as well as the relationship between the data and scripts and other data context information. Specifically, a metadata includes but not is limited to following properties:

- *Resources* describe and locate all packaged data. The descriptor could be in JSON or XML format while the paths could be a local path within a package (inline) or URLs pointing to remote storage (non-inline).
- *Scripts* indicate the location and purposes of data processing scripts on the datasets and specify the correlation among scripts and data. This property helps analysts specify what operations have been done on which datasets.

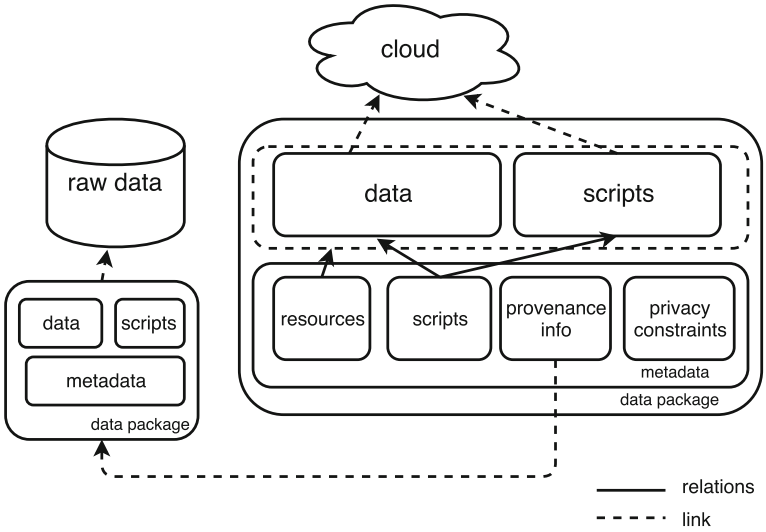

Fig. 3. Data package structure

- *Provenance information* is a sequence of links pointing to the previous data packages from which current package is generated. After acquiring a data package, analysts can modify the package content and create a new package. A *package chain* is formed when this activity is repeated. Analysts can trace the data usage back to the original dataset through the package chain.
- *Privacy constraints* record the privacy constraints imposed on the data in the package. When data providers apply privacy-enhancing techniques to generate anonymous data or expose their data partially, the operations they used to preserve data privacy are informed to analysts so that they can take corresponding tactics in their analysis.
- *Other descriptor* includes data schema, author, contributor, version, etc.

A simplified data package example in JSON is shown in Listing 1.3. The required properties are listed, others are omitted due to length limitation.

```
{       "name": "dataPackage",
        "id": "",
        "sources": [{
                "title":"hr-analytics dataset",
                "path": "https://www.example .com/datasets/hr-analytics"
        }],
        "resources": [{
                "path": "http:/www.example.com/hr-analytics.csv",
                "schema": "{...}"
        }],
        "scripts": [{
                "name": "retreive_good_employee_who_left",
                "path": "",
                "type": "python",
                "resources": ["good_employees_left", ".."]
```

```
}],
"provenanceLogs": [{
        "lastPackage": "",
        "created_in": "06/12/2017",
        "path": "http://example/HrAnalytics/dataPackage"
}],
"privacyLogs": [{
        "script_name": "",
        "description": ""
}]}
```

Listing 1.3. A data package example

In our data service architecture, the data package acts two roles as below:

Data Package as a Resource. As introduced in Sect. 4.1, the *packager* can package data into a non-inline resource by a path pointing to the remote storage. Apart from the link to the data, the scripts inside the package also record the user's exploration process. Data package can be created through POST and retrieved through GET. The included data, scripts, and metadata can be acquired, updated and deleted by the HTTP methods (GET, PUT and DELETE).

Data Package as a Context Service. Data package provides provenance information, the upstream lineage, and the data constraints like privacy compliance policy. For example, data publisher can use a random value perturbation techniques to hide sensitive data by randomly modifying the data values using additive noise while preserving the underlying probabilistic properties of the dataset so that a predictive analysis can be performed. The metadata in the data package describes this manipulation and other privacy constraints so that data consumers are more informed on the assumptions of data for later analysis.

By using the packager resource and created packages, data providers can package and share the processed data with the scripts applied to the data. Due to the data package, data consumers can be more informed what happened to the data, and take more effective actions to do further analysis without preparing data from scratch.

5 Evaluation

We conducted a case study to do a comparative evaluation of the proposed data service design against the OData REST design. OData is an OASIS standard protocol that defines how to build the RESTful APIs for open data. Our evaluation focused on three metrics including REST maturity, interoperability, and discoverability.

5.1 Case Study

To exercise our design and validate its feasibility, we selected one dataset and related kernels (data processing scripts) from Kaggle[12], which is a public platform for data analytics community to explore and produce predictive models on

[12] https://www.kaggle.com/datasets.

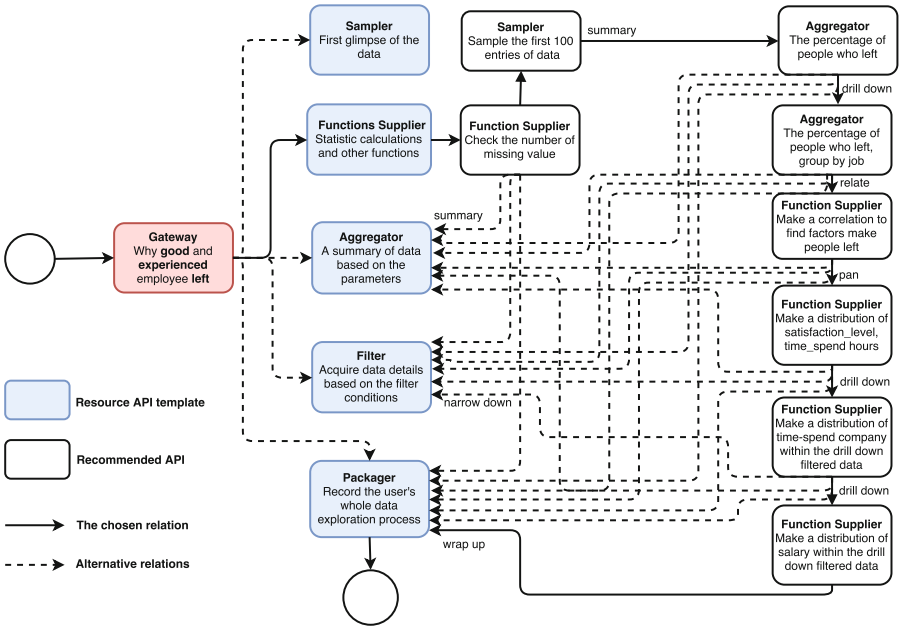

Fig. 4. HR analytics data exploration roadmap

the open datasets. The selected dataset is a Human Resource (HR) analytics dataset with 15000 rows.

Our RESTful APIs are implemented using Apache CXF JAX-RS in java. All the data returned are in JSON value. We simulated data exploration processes based on 8 published kernels on this HR dataset using our data service APIs.

Figure 4 shows the roadmap of one data retrieval process with the alternative relations and the resources. Based on the data exploration process, data analysts starts from *gateway* which responds with a list of resources to be selected. He starts his work from checking the data quality by the resource function supplier, which returns the numbers of missing values for the selected attributes, as well as links pointing to the *packager*, the *function supplier*, and the *aggregator*. Then the analyst sends a GET request to the *sampler* to retrieve a sample data, the returned response contains a defined size of data sample as well as links to other resources. Next, the analyst chooses *aggregator* to retrieve the summary information of the data.

In every step of the response, apart from a link pointing to the main resource API template, a specific API with predicted parameters will be recommended with an indicative `href` property. When the analyst makes a request of the percentage of left people, the response includes a *drill down* link pointing to a predicted *aggregator* API to group left people by their job. Afterward, the analyst can *pan* in the *function supplier* to gain other features of retrieved data until he is satisfied. Finally, the analyst moves to *filter* to retrieve the data of

the best and most experienced employee who have left. Alternatively, he can send a POST request to *packager* which can wrap up the data with his previous operations. The data package created can be shared on any data sharing platform for the purpose of reusing result and process of the data exploration.

5.2 Analysis

Maturity. We applied the Richardson Maturity Model[13] to evaluate how well our data service APIs adhere to REST principles. This Model categorizes a RESTful Web service into three levels of maturity according to the degree of its adherence to REST principle. Level 1 and level 2 specify resources and the HTTP methods respectively. The highest level uses HATEOAS to discover the next possible actions towards the clients.

Compared with OData service APIs which fails at level 3 because there is no guidance for services to include links or self-documentation in response. Our approach follows HATEOAS to provide links in the message body to trigger state transition in the client application. For instance, a GET operation on the gateway resource returns a response body with a list of all resources that can be of interest to start interacting with. Based on the navigation model, data service can navigate the users through resources and perform the user-desired operations using hyperlinks. Thus, our data service APIs achieve the highest level of maturity of REST.

Interoperability. Interoperability refers to the ability not only to exchange information (syntactic interoperability) between two systems via interface but also to correctly interpret data being exchanged (semantic interoperability). The important aspects of interoperability involve discoverability of services and handling of response from service requestor [13]. The Levels of Conceptual Interoperability Model (LCIM) defines five levels of interoperability maturity. The lowest level signifies systems that do not share data at all. The highest level indicates systems that work together seamlessly without mistakes interpreting each other communication [14].

Most OData REST APIs achieve syntactic interoperability inherently because they provide uniform, standard, and stateless interface on top of HTTP. However, their semantic interoperability is not guaranteed due to their simple message format without containing any context information. Table 1 compares the OData service design and our Data Service from three aspects, including analytical operation, analytics process, and context information shared in the analytics pipeline at semantic level.

Our proposed data service API design enables semantic interoperability of REST-based application and so partially reaches the highest level of LCIM for data analytics in following two points:

[13] https://martinfowler.com/articles/richardsonMaturityModel.html.

Table 1. Comparison of information sharing at semantic level

	OData service	Our design
Analytics operation	• Mapping to the under-lying data schema	• Conforming to data analytic operations
Analytics process	• User driven • Manually constructing	• Define analytical relations in HATEOAS • Intelligent recommendation
Context information	• Manually collected by data users	• Package chain • Record data exploration process

Interpretation of Analytics Domain Operation. With the help of the navigation mechanism, which semantically interprets the underlying interactions in analytics process, a resource pointed by another known resource can be discovered by the users. Further, the navigation model provides the request with predictive parameters targeting users'requirement. As described in Sect. 5.1, an aggregator API to group the percentage of left employee by job can be recommended for client who requested the percentage of left employee.

Sharing of Data Context Information. The data package has a rich, extensive, and self-descriptive structure. A data package containing all the context information of analytics pipeline ensures the data consumers and data publishers have a common view of the requested services and data. The data package clearly shows the provenance information about when and what has been done by whom on the provided data and privacy information determining how, when and to what extent information about the provided data will be released to data users.

Discoverability. Discoverability means that when a service consumer requests a resource, it receives URLs pointing to the resources associated with the current resource in the response message. HATEOAS enables the discoverability of web services. However, it is difficult to discover a service automatically without specifying the semantics of operations in the response. Discovering services in services using conventional REST design is time-consuming and error-prone.

Our data service proposes a roadmap of data exploration that defines different relations in links property so that a service can be reached from different resources. In addition, we use HTTP OPTIONS to inform users what operations and parameters can be performed on the resource. The proposed data service design partially fill the semantic gap of HATEOAS. Our HATEOAS with semantics enables auto-discovery for applications and presents the services as a graph illustrated in Fig. 2.

The Discoverability makes it possible to automate service interactions. Compared with the general-purpose search API like Twitter REST API, our data service APIs automatically provide developers with a list of available endpoints along with information on how to interact with the endpoints.

Reusability. Reusability is the degree to which a component can be used in multiple business process or applications, without much overhead on configuration and modification. Data package, as a media type, can support reusable data and processing scripts. For example, when we GET and run the scripts that the data package created from HR data exploration process, we can reproduce the previous data operations. Given the same dataset and deterministic query, the result is identical to the old one. This proves that the data package can be reused and shared among analysts with the same data exploration purpose on the same dataset. In addition, since the data package is a light-weight data container that packages links to diverse data source and metadata, it will not cause big performance burden for client environment. The flexible and extensive data structure of data package allows users to customize their package based on their specific requirement, which further improves the reusability of the data package.

5.3 Discussion

The case study demonstrates that our data service satisfies the first requirement in Sect. 3 by supporting the data exploration processes from Kaggle using Python or R library. Second, the case study shows that our data service satisfies the second requirement and achieves better interoperability and discoverability compared with the existing solution. Last, the case study shows that using data package is an effective way to reuse and share the derived data and processing scripts among analysts, so the third requirement is satisfied.

6 Conclusion and Future Work

This paper proposes a REST-based data service API design, which specifies data retrieval interface targeting data analytics. Our approach takes advantage of REST properties and its related hypermedia-driven features to make resource APIs generate and navigate each other automatically based on analytical needs. In addition, we introduce a mechanism to package data source, primitive operations, and data context together for users to customize and reuse the data exploration process. Our evaluation shows that this approach can enhance the interoperability and discoverability of data services and the reusage of data exploration processes. Our future plans include an extension of the approach to accommodate multiple data sources, enhance service discovery. We also plan to conduct more user studies.

References

1. Jagadish, H., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., Shahabi, C.: Big data and its technical challenges. Commun. ACM **57**(7), 86–94 (2014)
2. Khan, H.A., Sharaf, M.A., Albarrak, A.: Divide: efficient diversification for interactive data exploration. In: Proceedings of the 26th International Conference on Scientific and Statistical Database Management. ACM (2014)

3. Idreos, S., Papaemmanouil, O., Chaudhuri, S.: Overview of data exploration techniques. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 277–281. ACM (2015)
4. Borkar, V., Carey, M., Mangtani, N., McKinney, D., et al.: Xml data services. Int. J. Web Serv. Res. **3**(1), 85 (2006)
5. Dillon, S., Stahl, F., Vossen, G.: Towards the web in your pocket: curated data as a service. In: Nguyen, N., Trawiński, B., Katarzyniak, R., Jo, G.S. (eds.) Advanced Methods for Computational Collective Intelligence. Studies in Computational Intelligence, vol. 457, pp. 25–34. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-34300-1_3
6. Fielding, R.T.: Architectural styles and the design of network-based software architectures. University of California, Irvine Doctoral Dissertation (2000)
7. Wilde, E., Pautasso, C.: REST: From Research to Practice. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-8303-9
8. Page, K.R., De Roure, D.C., Martinez, K.: Rest and linked data: a match made for domain driven development. In: Proceedings of the Second International Workshop on RESTful Design, pp. 22–25. ACM (2011)
9. Robinson, I.: RESTful Domain Application Protocols, pp. 61–91. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-8303-9_3
10. Richardson, L., Amundsen, M., Ruby, S.: RESTful Web APIs: Services for a Changing World. O'Reilly Media, Inc. (2013)
11. Hernández, A.G., García, M.N.M.: Metadata Architecture in RESTful Design, pp. 459–471. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-8303-9_21
12. Hellerstein, J.M., Sreekanti, V., Gonzalez, J.E., Dalton, J., Dey, A., Nag, S., Ramachandran, K., Arora, S., Bhattacharyya, A., Das, S., et al.: Ground: A data context service. In: CIDR (2017)
13. Clements, P.C.: Software architecture in practice. Diss. Software Engineering Institute (2002)
14. Tolk, A., Muguira, J.A.: The levels of conceptual interoperability model. In: Proceedings of the 2003 Fall Simulation Interoperability Workshop, vol. 7, pp. 1–11. Citeseer (2003)