Design RESTful Data Service for Data Analytics

Yun Zhang*†, Liming Zhu*†, Xiwei Xu*†, Shiping Chen*†, An Binh Tran*
*Data61, CSIRO, Sydney, Australia
Email:{Firstname.Lastname}@data61.csrio.au
†School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

***Abstract*—Data service APIs provide a uniform and filtered access to data. However, existing *one-size-fits-all* API design for data services is often inefficient for data analytics driven data retrieval because it focuses on the underlying data sources rather than the data analytics needs. The *one-off* API representations lack support for different patterns of data retrieval, and do not leverage hypermedia and data packaging techniques used in RESTful API design. Besides, current data service APIs are designed and configured statically rather than generated dynamically based on data analytics needs in different contexts. Lastly, context information about the origin, scope, and usage of the data provided is seldom considered in data service design.**

In this paper, we propose an analytics-focused API design for data services. First, a set of RESTful conversation models are introduced to depict the interactions between data consumers and data sources for different data retrieval patterns. Second, following HATEOAS principle of REST, a navigational model is designed to enable dynamic generation of APIs and navigation among them. Finally, data package technique is extended to facilitate efficient and flexible data retrieval. We evaluate our approach using a case study and make a comparison analysis between our approach and conventional DaaS (Data as a Service) RESTful APIs. The evaluation shows that our approach is more efficient and effective for data retrieval in data analytics.

***Index Terms*—Data analytics; Data service; REST; Data package**

I. Introduction

A large amount of data has been published on the web for the last decades, which brings great analytic values across both industries and academia. For example, in Twitter, more than 1 million tweets are published every 10 minutes. Another example is government open data platforms, like data.gov.au [1], which already shares over 20 thousands of datasets for free. How to properly and efficiently retrieve these data for data analytics is becoming a hot issue [22] [28].

To analyse a big 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 [23]. Data exploration, which is a 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 [21]. The other two commonly used data retrieval patterns are batch and stream retrieval. In batch retrieval, data is collected to do some relatively time consuming data analytics tasks. In stream retrieval, a query is being repeatedly executed over a stream of data at real-time. Thus, rather than batch processing, stream retrieval pattern allows users to analyze the data-in-motion.

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. The 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 unified, scalable and filtered interfaces for data analysts to retrieve data [14]. Many companies and platforms, like Twitter, Google, and CKAN [2] offer data service APIs that provide simple and easy-to-use access to some of their resources. Data services allow third-parties to easlily 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 *one-size-fit-all* 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 [16]. Second, the *one-off* data queries are isolated and static, which does not leverage the HATEOAS (Hypermedia as the Engine of Application State) principles of REST. It does not support service self-discovery. Third, there is no standard mechanism to provide context information about the origin, scope, and usage of the data behind 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 analytics's need. Our contributions include (1) a set of RESTful conversation models for the interaction between data consumers and resources, (2) a navigational data model to help discover and generate data service API dynamically, and (3) a mechanism using extended data package to contain, publish and share data processing scripts and data context information. We evaluated the proposed API design through a feasible case study and discussion about quality attributes. The evaluation shows that our approach....

The rest of this paper is organized as follows. Section II describes some related work. Section III introduces the data service design including the three conversation models, the

navigational model and the extension of data package. Section IV uses a case study to evaluate our API design. Section V concludes the paper.

II. Background and Related Work

REST (REpresentational State Transfer) [17] is an architecture style for designing web applications. Following REST design principles, data service is identified by URI as a resource, and interacted with client applications by request-response messages. Protocols and Structures for Inference (PSI) [3] specifies a RESTful architecture for presenting machine learning and related moudules as RESTful web services, whereby the data access 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 commercialized databases, like Amazon SimpleDB [4] and Microsoft SQL Azure [5], provide accessible data service APIs to their data stores. ODBAPI [27] is a uniform REST API to execute CRUD (create, read, update, delete) operations across various relational and NoSQL data stores. HTSQL [6] enables accessing SQLServer via HTTP arbitrarily, which is an advanced query language on the web designed for data analysts who have complex inquiries across relational databases. However, these CRUD-based data services are designed and implemented according to database schemas and pre-assembled indexes without reference to the underlying domain application protocol.

In order to build a domain application protocol over HTTP which is domain agnostic in the web application, some additional, explicit semantics are needed [30]. 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 [24].

Hypermedia as the Engine of Application State (HATEOAS) which is the hypermedia constraint, requires that the service should embed links in its responses. The links represent the next possible actions that the client can take [17]. However, there is a semantic gap for clients to be navigated by hyperlink automatically. In order 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 [26]. AtomPub is designed to cover all the collection-based APIs but cannot reflect different application semantics. Microsofts Open Data Protocol (OData) [9] derived from AtomPub defines the protocol semantics for filtering and sorting a collection of data, using a query language similar to SQL.

However, the semantics of the relationship between resources focus on data instead of analytics operations, and the related metadata services only present limited description documents [25]. Our approach fills this semantic gap by specifying the domain application protocol of data exploration in analytics and bringing this protocol into the HATEOAS implementation of our services.

Metadata provides self describing information about web resources in REST architecture, which enable automatic processing of web resources [20]. However, the metadata is sent in headers of HTTP message and restricted to provide information about the syntax used in the resource representation. In addition, the semantics about the origin, scope, and usage of the data is less considered. Ground [19] 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. 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 package, which guide 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.

III. Data Service Scenario

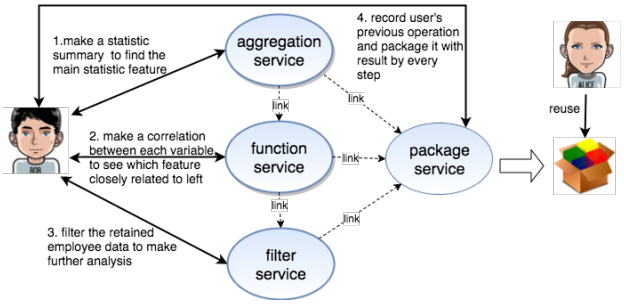

Fig. 1. Data Exploration Scenario Using Data Service

We use a practical human resource analytics scenario to show how data services aid in data exploration and facilitate collaboration. The purpose of the analysis is to help a company understand why some of their best and most experienced employees are leaving prematurely and predict who will leave in future.

Analyst Bob first makes a request of statistic summary of the data provided by the company, investigating the sum, average, min, max and medium of numeric attributes respectively. Apart from the result of aggregation, the data service also returns the URIs of services for all the possible next steps. Bob then selects the function service to make 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 uses the filter 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 have left the company. After developing insight from the data, Bob uses these data to conduct an analysis model to predict who will leave. After

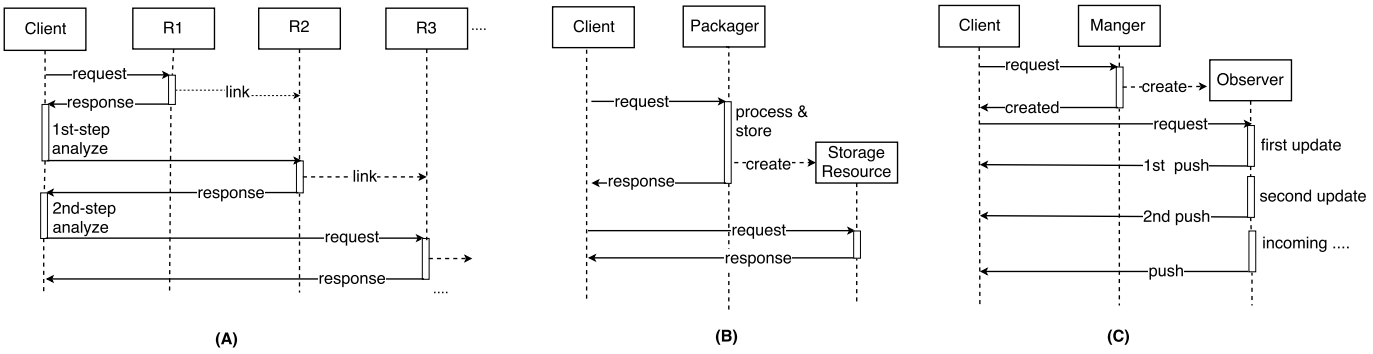

Fig. 2. Conversation Models

Bob finishes explorative analysis, he invoke package service to wrap his data exploration process as a data package and share it on the web. The data package is then discovered by Alice to do further analytics without preparing data from scratch. Alice can also reprocess the operations included in the package to verify Bobs exploration result. Even, she can leverage Bob's explorative analysis operation referred service to extend and constrct her own operation. This style of work could be repeated, by producing and sharing Alice's data package for another analyst.

There are three main benefits from using the data service to realize data exploration. First, from the data consumers (like Bob and Alice) perspective, pushing computation to the data is more efficient than pulling the whole data out for computation. Therefore, it is desirable to move the basic and frequently used analytic operations to the data sources and wrap them as services. The resources interconnected via hyperlinks could navigate analysts to find the data they need quickly and preciously. Second, from the data publishers (like the company) perspective, they can expose lightweight data features for analytics to protect data privacy and commercial interests rather than open the whole data. Lastly, the data package enables data consumers to share the result of their exploration as data publishers, and thus brings the benefit of reusability, flexibility and customizability.

IV. Service Design

A. Conversation Models

RESTful conversation [18] is a sequence of interactions between a client and resources. The conversation route is controlled by the resources following the HATEOAS principle and driven by the client. Three RESTful conversation models of the data retrieval patterns for data analytics are depicted by UML sequence diagrams.

1) Interactive Conversation Model: In order to reduce the computation cost and improve the timeline of analysis, the interactive analytics involves the human insight into the process of data analysis. As shown in Fig.2(A), a *client* issues the first query to the resource R1, the response of R1 contains the requested data and the possible links that the *client* could select. The *client* analyzes the returned result, and then chooses the link pointing to the resource R2. Again, this conversation repeats until the result is satisfied.

2) Batch Conversation Model: Batch retrieval means a group of operations are executed for retreiving data in batch. As the task might be time-consuming, the request can be processed asynchronously while a link pointing to the dataset is returned immediately to the client. The main interactions of batch conversation model are shown in Fig.2(B), a *client* makes a request to the resource *packager* which is responsible for extracting data from the data source. *Packager* generates the data package and stores it into the remote storage, meanwhile responding instantly with a hyperlink referring to the to-be storage resource. The *client* keeps polling the data status from the *storage resource*, until the data is ready.

3) Streaming Conversation Model: In the streaming data retrieval, a client can receive the real-time updates or events occurred from the resource. This pattern allows analysts to analyze the data-in-motion. The streaming conversation model involves three roles: the client, the manager and the observer. Specifically, as shown in Fig.2(C), the *client* makes a request on the resource *manager* with parameters like window, interval or operations in the representation. The *manager* identifies the *client*s interest, creates the *observer* based on requirement of the *client*, and then responds the link of the *observer*. Afterwards, the *observer* monitors the state of the data and delivers the user-defined stream data and push-like notification in real time.

B. Service Architecture

In this section, we introduce the service architecture with the main resources. As shown in Fig.3, the main components in our architecture are the *gateway, filter, aggregator, sampler, function supplier* and *packager*. We propose a navigational model to describe the relationship of the main resources in the domain of data analytics and motivate the idea to use this model as a modeling tool for REST APIs. We apply data package technique into our data services and showcase its usage and advantages in a data analytics pipeline environment.

1) Key Resources: **Gateway** is a self-describing resource that exposes the metadata defining data schema and other information like data type (for example, dynamic, static), data size, attributes and description about this dataset, which helps

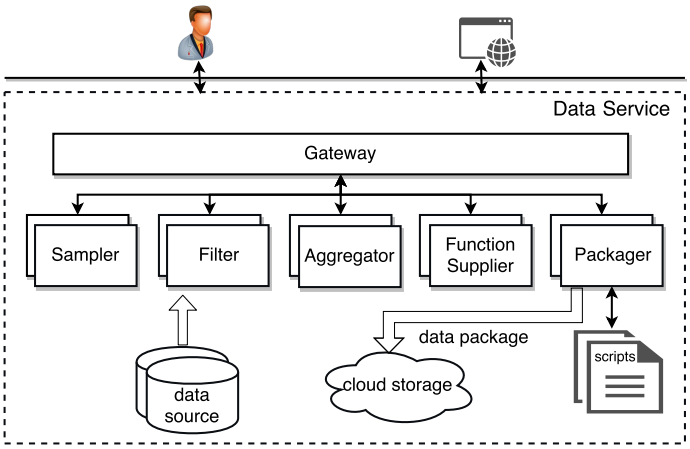

Fig. 3. Data Service architecture

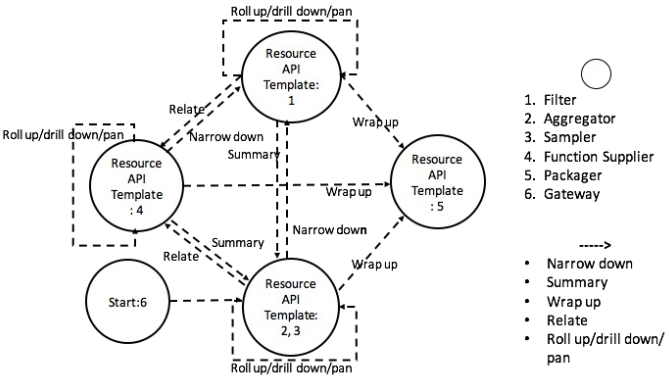

Fig. 4. Navigational Model

the data analyst have an initial understanding about the dataset. Further, the *gateway* acts as a role of discovering related resources about the dataset.

Filter filters the data 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. Dimensions are divided into atomic and composite dimensions according to the data features. The composite dimension is hierarchical, for example, a time dimension is composed of year which consists of four quarters containing all months. On the other hand, the measures are not only based on the built-in query schema but also can be a combination of measures expressed in mathematic operation as well as statistics functions.

Aggregator allows users to have a whole characteristic of the large subset of the data without extracting specific data items. It can perform aggregation function over one attributes values and group by the attributes name. The aggregation queries have constraints based on the attributes numeric or non-numeric features.

Sampler provides diverse sampling methods, like random sampling, to sample a specified size of data with or without placement, which allow analysts to quickly build and test their models within a sample of data that can fit into the clients memory. The sample size and specific sample methods are defined by users.

Function Supplier provides a set of functions, which effectively assist data analysts as auxiliary means. For example, *correlation* is for understanding the relationship between attributes; *isnull* is a necessary function for data cleansing, which is used to check for the missing value to help analysts know the data quality. Then analysts can take three methods for missing values in our data services: drop the record with missing values, drop variables and fill in the missing value with a default or user-defined value. All these methods can be supplied by the *filter* while retrieving data.

Packager retrieves the big volume data, wraps them with optional primitive operations which are presented as scripts into package, and then store them in external storage. Using the packager, the data consumer can acquire a targeted subset of 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 discuss more detail as below.

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

The navigational model is shown in Fig.4. The circles represent the resources introduced in last section, while arrows correspond to the relationships between the resource API templates. The relationships across resources are categorized into four types: *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. 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 navigational 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 navigational 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 or attributes 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 where and having 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 measure. *Pan* allows users to change the angle they observed by changing the dimensions of data or the operations used.

The navigational model incorporates the analytics domain semantics into HATEOAS. Specifically, the `links` property is used to represent all the related actions and resources for each resource with which clients can interact. `links` is defined as an array of Linked Description Objects(LDOs) in JSON Hyper-Schema [11], which obeys HATEOAS constraint and assist discover 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 looking if the representation has embedded `links` to the next resource.

The navigational model involves in both dynamical discovery and generation, which enable discover the resource API template driven by client while generate the API parameters based on the previous input from client. We present their schemas of Links separately in listing 1 and listing 2.

As shown in listing 1, each `links` comprises of `rel` which present the meaning of related action, and `href` which points to the location of resource. The `rel` indicates *relate, summary, narrow down* and *wrap up* for resource APIs' dynamic discovery. The `method` and `schema` properties specify the HTTP method and data format for the input. Afterwards, client can send a HTTP OPTIONS request to acquire further assistance on how to form a specific API. Listing 2 defines the schema of `links` for generating the specific resource APIs. According to the different semantics of `rel`, the new parameters are intelligently generated based on the measures and dimensions in the `base`, and form a new resource API as a `href` property for client. The `rel` indicates *roll up, drill down* and *pan* for resource APIs dynamic generation.

```
{
        "$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. 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": [{
                "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 2. HyperSchema of Links for Dynamical Generation

3) Data Package: We adopt data package as a media type to allow users to customize and reuse the data exploration process to support more flexible and efficient data retrieval. By using the packager resource, data providers can package the processed data with related scripts. For example, assume that the World Bank publishes a dataset about middle income countries together with a script that calculates the middle income based on annual analysis. Thus, the data analyst can effectively query data and perform analysis based on the scripts included in the data package.

Fig.5 gives an overview of the extended data package, which may contain (1) data such as tables and files stored remotely in cloud storage or 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 contents of the package, as well as the relationship between the data and scripts and the data context information. Specifically, a basic metadata consists of the following properties:

- *Resources* describe and locate all packaged data. The descriptor could be in JSON or XML format while the

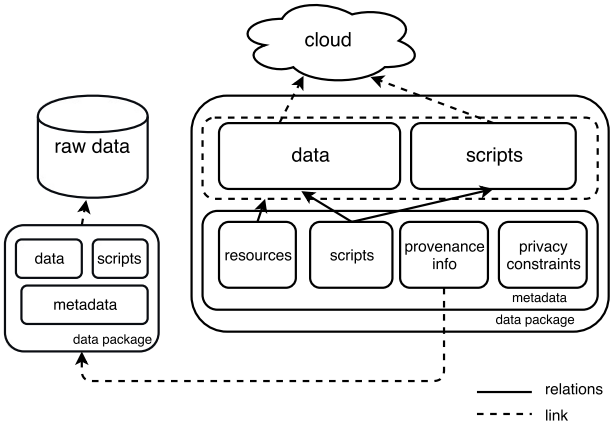

Fig. 5. Data Package Structure

paths could be a local path within package(inline) or URLs pointing to remote storage(non-inline).

- *Scripts* indicate the location of inline and non-inline scripts and specify the purpose of the scripts on the data sources and the correlation among scripts and data. This property helps analyst understand what operations have been done on which datasets.
- *Provenance information* is a sequence of links pointing to the previous data packages from which current package is generated. It records the package chain linked by the hypermedia to allow users to trace the data usage in the data pipeline.
- *Privacy constraints* record the privacy constraints imposed on the data of current package. When data providers apply privacy-enhancing techniques to generate anonymous data or expose their data partially, their operations could be 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 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": "06/06/2017",
                "path": "http://example/HrAnalytics/
     dataPackage"
        }],
"privacyLogs": [{
                "script_name": "",
                "description": ""
        }]
}
```

Listing 3. A Data Package Example

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

Data package as a resource As introduced in batch conversation model and the key resources section, the packager can package a big volume set of data into a non-inline resource by a path pointing to the remoted storage. Apart from the link to the data, the scripts inside the package that record the users exploration process. Data package can be created through POST and be retrieved through GET. The included data, scripts, and metadata respond to GET, PUT and DELETE respectively.

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 privacy constraints so that data consumers are more informed on the assumptions of data for later analysis.

V. Evaluation

We implement a case study to conduct a comparative evaluation of the proposed data service design for analytics against the traditional DaaS REST APIs with respect to REST maturity, interoperability and self-discoverability.

A. Case Study

We selected one open dataset and its related data exploration process from Kaggle [12] to implement the HR analytics scenario discussed in Section III. The RESTful web services are implemented using Apache CXF JAX-RS in java. Figure.6 shows the roadmap of the data retrieval process with the alternative relations and the resources. During the interactive data exploration, each service is automatically discovered and related parameters are intelligently generated. The process can be navigated to the packager for executing multiple operations in batch.

1) Interactive Process: The analyst 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 indicating the *filter*, the *aggregator*, the *sampler* and the *packager*. Then the analyst sends a GET request to the *sampler* to retrieve a sample data, the returned response contains a defined size of data

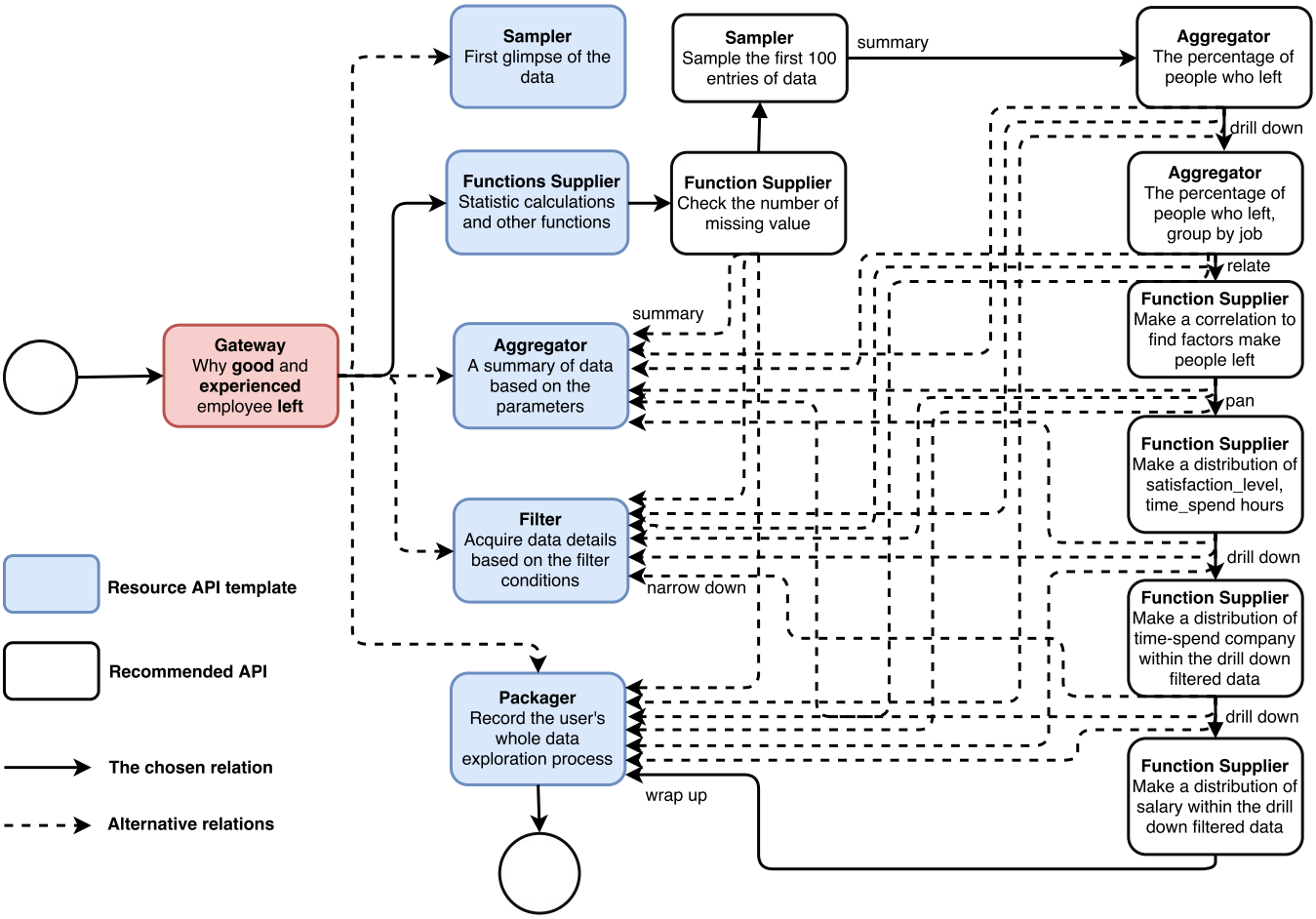

Fig. 6. HR Analytics Data Exploration Roadmap

sample and the URIs of services for all the possible next steps. Next, the analyst chooses the *aggregator* to retrieve the summary information of the data. In each 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. As shown in the Fig.6, when the analyst makes a request of percentage of left people, the response includes a *drill down* link pointing to a predicted *aggregator* API to group left people by their job. Afterwards, he selects the *function supplier* to discover a correlation between each pair of attributes. The results show that on average, employees who left the company have lower satisfaction levels. The returned result guides the analyst to query the distribution of each feature correlated with the left employees. After knowing all the features of employees, the analyst moves to the *filter* 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 have left the company. Fig.7 demonstrate the detailed RESTful interactions of data exploration we described above.

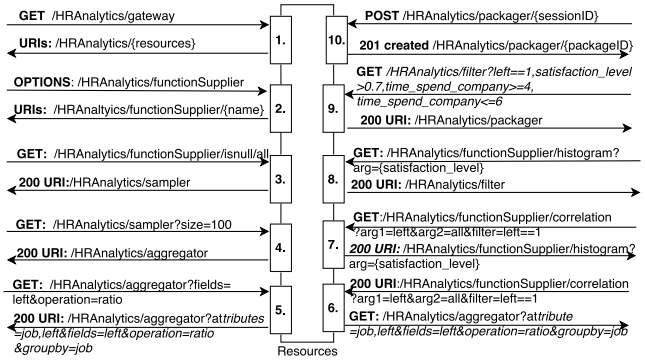

Fig. 7. Interactions in the Implementation of the Case Study

2) Batch Process: After the analyst finishes his exploring the data, as shown in Fig.6, his data exploration process can be wrapped as a script into data package. He can share the data package in any data sharing platform for the purpose of reusability of data exploration.

The data package is then discovered by another analyst to do further analysis without preparing data from scratch.

Based on the context information in the package, the second analyst can reprocess the scripts to acquire the initial analyst's exploration result. The content of the POST request is shown in listing 4. The packager will execute the operations in batch in the backend, and return the URI of to-be prepared result. Afterwards, the second analyst will GET the result until the result is ready.

```
POST /HRAnalytics.com/packager
Content-Type: application/json
{
        "script":[{
        "id": "http://localhost:8080/HRAnalytics/
     functionSupplier/isnull/all", "operation":"GET"
},{
        "id": "http://localhost:8080/HRAnalytics/sampler?
     size=100",
         "operation":"GET"},{...}
  ]}

response: 201 created /HRAnalytics/packager/{id}

GET http://localhost:8080/HRAnalytics/HRAnalytics/packager
     /{id}

//result is not ready
response: 202 Accepted Content-Type: application/json
{
        "completed": false
}

//result is ready
response: 200 0k Content-Type: application/json
{
        "completed": true,
        "results": [
{
        "id": "http://localhost:8080/HRAnalytics/
     functionSupplier/isnull/all",
        "output": "http://localhost:8080/hranalytics/
     packager/{id}/output_1.json"
        },{
        "id": "http://localhost:8080/hranalytics/sampler?
     size=100",
        "output": "http://localhost:8080/hranalytics/
     packager/{id}/output_2.json"
        },{...}
]}
```

Listing 4. Asynchronous Interaction with Packager

B. Evaluation

TABLE I
COMPARISON OF INFORMATION SHARING AT SEMANTIC LEVEL

	DaaS	RESTful Data Service
Analytics operation	Mapping to the underlying data schema	Conform to data retrieval patterns
Analytics process	• User driven • Manually input	• Define anlytical correlations in HATEOAS • Intelligent recommendation
Context-info	• Manually collected by data users	• Package chain • Record data exploration process

1) Maturity: We employed the Richardson Maturity Model [13] evaluate how well our data service API adheres to REST principles. This Model categorizes a Web service into three levels according to the degree of its adherence to REST principle. The highest level uses HATEOAS to present the next possible actions towards the clients.

Compared with DaaS REST API which fails level 3 because there is no links or self-documentation in response, we implemented HATEOAS to supply links in the message body that trigger state transition in the client application. For instance, a GET operation on our gateway resource returns a response body with a list of all resources that can be of interest to start interacting with. Given our navigational model, data service can navigate through data and perform the user-desired operations using hyperlinks. Thus, our data service API achieves the highest level of maturity of REST.

2) 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 [15]. 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 [29].

Most DaaS REST API satisfies syntactic interoperability inherently because it provides uniform, standard, and stateless interface on top of HTTP. However, their semantic interoperability is not guaranteed due to its simple message format without containing any context information. Table I compares traditional DaaS REST API and our Data Service API from analytical 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:

Interpret analytics domain interoperation 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. Further, the navigational model processes the request with predictive parameters targeting users requirement. As described in the use case, an aggregator API with additional parameters to group the percentage of left employee by job is recommended for client who requested the percentage of left employee.

Share 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 understanding of meaning 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.

3) Self-discoverability: Discoverability means that when a service consumer requests a resource, it receives URLs to

find 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 because the service does not specify the semantics of operations in the response. Therefore, discovering services is time consuming and error pone.

Our data service proposes a roadmap of data exploration that define different relations in links property, so that the service can be reached from different resources.In addition, we facilitate HTTP OPTIONS to inform user what operations and parameters can be performed on the resource. The proposed data service design partially resolve semantic gap in the HATEOAS implementation for data analytics. Our HATEOAS enabled application supports auto-discovery and presents the services as a graph illustrated in Fig.3.

The self-discoverability makes it possible to automate service interactions. Compared to the general-purpose search API like Twitter REST API, our data service APIs are automatically generated by providing developers with a list of available endpoints along with some information on how to interact with that endpoint, that save the cost of development time and effort.

VI. Discussion

There are three main benefits from using our data service to realize data exploration. First, pushing computation to the data is more efficient than pulling the whole data out for computation. Second, from the data publishers' perspective, they can expose lightweight data features for analytics to protect data privacy and commercial interests rather than open the whole data. Finally, the data package enables data consumers to share the result of their exploration as data publishers, and thus brings the benefit of reusability, flexibility and customizability.

Our current service architecture covers the main resources involved in the interactive and batch conversation models. Regarding the streaming conversation model, a unified web streaming data API adhering to REST will be developed as a resource in our future work. We consider two options to realize a REST streaming API. One is to delegate streaming process to the REST web service while the other one is to stream data via HTTP. We plan to study analysis requirements for streaming data, and adopt an approach that best suits analytics processes.

Our data services are designed based on the single data source. It is also suitable for multiple data sources provided that a common semantic data model integrates the data across heterogonous sources. Our future work will focus on how to create this semantic data model targeting data analytics as a generic resource model for data service.

VII. Conclusion

Data service provides a filtered interface between data sources and data analysts. However, the existing data services are designed from the perspective of data publisher, which highly limited the accessibility of data resource. Moreover, due to its *one-size-fit-all* characteristic, the data-centric web service cannot meet the requirement of data analytics regarding interactive, stream and batch data retrieval patterns. In order to fill the gap, we proposed a data service API design focused on data analytics. The proposed approach takes advantage of REST to navigate and provide different resource APIs according to the different requirement and context. Besides, we provided a guidance for packaging data source and primitive operations with different characteristics for more efficient data retrieval. The design has been evaluated by a practical case study, and the result showed that our data service can assit retrieve data for analytics effectively and efficiently.

References

[1] http://data.gov.au/. (last verified September2, 2017).

[2] ckan,https://ckan.org/. (last verified September4, 2017).

[3] Protocols and structures for inference.http://psi.cecs.anu.edu.au/. (last verified September2, 2017).

[4] AWS — Amazon SimpleDB Simple Database Service, http://aws.amazon.com/simpledb/. (last verified September2, 2017).

[5] Microsoft Azure.http://www.microsoft.com/azure/sql.mspx. (last verified September2, 2017).

[6] A Database Query Language-HTSQL, http://htsql.org/.. (last verified September2, 2017).

[7] Joe Gregorio and Bill de hOra (eds). The Atom Publishing Protocol. 2007,https://bitworking.org/projects/atom/rfc5023.html. (last verified September5, 2017).

[8] OpenSearch,http://www.opensearch.org/Home. (last verified September5, 2017).

[9] OData, http://www.odata.org/. (last verified September2, 2017).

[10] Data Package-Frictionless Data Specifications, https://specs.frictionlessdata.io/data-package/.. (last verified September2, 2017).

[11] JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON, http://json-schema.org/latest/json-schema-hypermedia.html. (last verified September2, 2017).

[12] Datasets — Kaggle,https://www.kaggle.com/datasets. (last verified September2, 2017).

[13] Fowler, M.: Richardson Maturity Model, https://martinfowler.com/articles/richardsonMaturityModel.html. (last verified September2, 2017).

[14] Vinayak Borkar, Michael Carey, Nitin Mangtani, Denny McKinney, et al. Xml data services. *International Journal of Web Services Research*, 3(1):85, 2006.

[15] Paul C Clements. Software architecture in practice. *Diss. Software Engineering Institute*, 2002.

[16] Stuart Dillon, Florian Stahl, and Gottfried Vossen. Towards the web in your pocket: Curated data as a service., 2013.

[17] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[18] Florian Haupt, Frank Leymann, and Cesare Pautasso. A conversation based approach for modeling rest apis. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, pages 165–174. IEEE, 2015.

[19] Joseph M Hellerstein, Vikram Sreekanti, Joseph E Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, et al. Ground: A data context service. In *CIDR*, 2017.

[20] Antonio Garrote Hernández and María N. Moreno García. *Metadata Architecture in RESTful Design*, pages 459–471. Springer New York, New York, NY, 2011.

[21] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 277–281. ACM, 2015.

[22] HV Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[23] Hina A Khan, Mohamed A Sharaf, and Abdullah Albarrak. Divide: efficient diversification for interactive data exploration. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, page 15. ACM, 2014.

[24] Kevin R Page, David C De Roure, and Kirk Martinez. Rest and linked data: a match made for domain driven development? In *Proceedings of the Second International Workshop on RESTful Design*, pages 22–25. ACM, 2011.

[25] Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs: Services for a Changing World*. " O'Reilly Media, Inc.", 2013.

[26] Ian Robinson. *RESTful Domain Application Protocols*, pages 61–91. Springer New York, New York, NY, 2011.

[27] Rami Sellami, Sami Bhiri, and Bruno Defude. Odbapi: a unified rest api for relational and nosql data stores. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 653–660. IEEE, 2014.

[28] Ramesh Sharda, Dursun Delen, Efraim Turban, Janine Aronson, and Ting Peng Liang. *Businesss Intelligence and Analytics: Systems for Decision Support-(Required)*. Prentice Hall London, 2014.

[29] Andreas Tolk and James A Muguira. The levels of conceptual interoperability model. In *Proceedings of the 2003 fall simulation interoperability workshop*, volume 7, pages 1–11. Citeseer, 2003.

[30] Erik Wilde and Cesare Pautasso. *REST: from research to practice*. Springer Science & Business Media, 2011.