Data service API design for data analytics

Author:
Zhang, Yun

Publication Date:
2019

DOI:
https://doi.org/10.26190/unsworks/21317

License:
https://creativecommons.org/licenses/by-nc-nd/3.0/au/
Link to license to see what you are allowed to do with this resource.

Downloaded from http://hdl.handle.net/1959.4/62990 in https://unswworks.unsw.edu.au on 2022-06-17
Abstract

Over recent years, Data as a Service (DaaS) has emerged as a new paradigm in the cloud computing environment. DaaS enables data provision as a service and provides controlled access to this data through RESTful APIs. Across the entire data analytics lifecycle, data service can be regarded as a method for data retrieval and exploration. However, existing data services fall short on supporting data retrieval and exploration for data analytics, as most of them are designed to simply query data based on underlying data schema rather than being driven by data analytics. Moreover, the current data service API representations only allow analysts to make one-off queries and do not provide them with any guidance on continuously exploring data. Last but not least, current data services do not support the reuse of data exploration processes and the data derived from data analysts.

Accordingly, to fill the gap discussed above, this research proposes Data Exploration as a Service (DEaaS) along with the data service architecture and RESTful API design to make data retrieval efficient and effective for data analytics. The contributions of the present research include:

- A set of RESTful conversation models for depicting data retrieval patterns;
- A RESTful data service architecture, incorporating an abstract data model and formalized resource design for data retrieval for data analytics;
- A navigation model to facilitate the dynamic discovery and recommendation
of service resources by enriching HATEOAS semantics and leveraging API call history;

- An extension of the data package incorporating data processing scripts and data context information, enabling the reuse of the data exploration process and the derived data.

Using a prototype implementation of DEaaS, case studies and experiments were conducted in order to carry out a comparative evaluation of the proposed data service design against a conventional data service design, named OData. The case study results show that the DEaaS approach has advantages over traditional data services in terms of usability, maturity, interoperability, discoverability, reusability, and adaptability. Moreover, the experimental results show that the proposed resource navigation approach can make DEaaS outperform existing data services in data exploration.
Acknowledgments

I would like to express my sincere gratitude to my principal supervisor, Dr. Liming Zhu, for his invaluable and inspiring guidance throughout the entire period of my Ph.D. studies. I am also thankful for Dr. Xiwei Xu, for her constructive feedback, and for her important and helpful support throughout my Ph.D. studies.

My special thanks go to my Ph.D. peers and Data61 colleagues, particularly Dr. Shiping Chen, Dr. Suhrid Sytal and An Binh Tran for their professional advice, technical help and assisting me to increase my domain knowledge.

From the bottom of my heart, I would like to express my deepest thanks to my parents, my husband, and my daughter, for their love, encouragement, and sacrifices. Without their support, this work would not have been possible. To them dedicate this thesis.
List of Publications


<table>
<thead>
<tr>
<th>Table 1: Mapping Between Chapters and Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter</td>
</tr>
<tr>
<td>Chapter 1 Introduction</td>
</tr>
<tr>
<td>Chapter 2 Background</td>
</tr>
<tr>
<td>Chapter 3 Methodology</td>
</tr>
<tr>
<td>Chapter 4 DEaaS Design</td>
</tr>
<tr>
<td>Chapter 5 Evaluation</td>
</tr>
<tr>
<td>Chapter 6 Conclusion</td>
</tr>
</tbody>
</table>
Contents

List of Publications v

List of Figures xi

List of Tables xii

1 Introduction 2

1.1 Background ...................................................... 2
1.2 Problem Statements .............................................. 4
1.3 Motivating example ............................................. 6
1.4 Requirements ................................................... 7
1.5 Research Questions ............................................. 8
1.6 Research Contribution ......................................... 10
1.7 Thesis Outline .................................................. 12

2 Background and Related Work 13

2.1 Data Preparation for Data Analytics ......................... 13
  2.1.1 Data Analytics ............................................. 14
  2.1.2 Data Retrieval ............................................. 17
  2.1.3 Data Exploration ........................................... 17
  2.1.4 Data Integration ........................................... 20
2.1.5 State of the Practice .................................. 22
2.2 Data Service for Data Analytics ...................... 25
  2.2.1 REST and HATEOAS ............................... 25
  2.2.2 Data as a Service ................................ 26
  2.2.3 DaaS for Data Retrieval Patterns ................ 30
  2.2.4 Service Discovery and Recommendation .......... 34
  2.2.5 Metadata Service ................................ 34
  2.2.6 Other Related Work ................................ 35
2.3 Summary and Conclusion .............................. 36

3 Methodology ............................................ 38
  3.1 Research Question Formulation ....................... 40
  3.2 Methodology for Deriving Research Solution ......... 41
    3.2.1 Literature Review ................................ 41
    3.2.2 Research Solution Development .................. 41
  3.3 Methodology for Evaluation ......................... 42
    3.3.1 Case Study ........................................ 42
    3.3.2 Experiment Methodology ......................... 44
    3.3.3 Conclusion ........................................ 46

4 Data Service Design for Data Analytics ............... 47
  4.1 Conversation Models ................................ 48
  4.2 Service Architecture ................................ 53
    4.2.1 Overview of Service Architecture ............... 53
    4.2.2 Data Model ....................................... 55
    4.2.3 Service Resources ............................... 56
  4.3 Resource Navigation ................................ 59
List of Figures

1.1 DaaS versus DEaaS ........................................ 4
1.2 Data Exploration Scenario Using Data Service ............... 6

2.1 Framework of the Literature Review .......................... 14
2.2 Data Analytics Lifecycle .................................... 15
2.3 Data Analytics Stack ........................................ 22
2.4 Partial Response Example .................................... 31
2.5 pagination Example ........................................... 31

3.1 Overview of the research methodology ......................... 39

4.1 Interactive Conversation Model ............................... 51
4.2 Batch Conversation Model .................................... 52
4.3 Stream Conversation Model ................................... 52
4.4 Overall design of DEaaS architecture ......................... 53
4.5 Navigation Model .............................................. 61
4.6 Update and search for a discovery tree ....................... 65
4.7 Data Package Structure ....................................... 75
4.8 Data Package Semantic Protocol .............................. 78

5.1 API Framework of DEaaS ...................................... 83
5.2 Repository Layer Classes ..................................... 84
List of Tables

1 Mapping Between Chapters and Publications ............... v

4.1 Definition of Resource Model Symbols .................. 57

5.1 Data Collection Procedure .............................. 93

5.2 Comparison of information sharing at Semantic level ......... 100
Chapter 1

Introduction

In this chapter, Section 1.1 provides the background of this research. Section 1.2 presents the problems motivating this research. Section 1.5 proposes research questions to address the problem statements. Section 1.6 discusses the main contributions of this research and briefly describes the research methods to be applied. At the end of the chapter, Section 1.7 presents the overall structure of this thesis.

1.1 Background

large amounts of data are increasingly being published on the web. For example, on Twitter, more than three million tweets are published every 10 minutes\(^1\). Another example is the open data platforms provided by governments, such as data.gov.au\(^2\), which already published over twenty thousand of datasets for free. The question of how to properly and efficiently retrieve these data for data analytics is becoming a hot topic [44].

Data analytics, by definition, is the activity of collecting, parsing and analyzing

\(^1\)http://www.internetlivestats.com/twitter-statistics/
\(^2\)https://data.gov.au
data to discover the patterns and features of this data in order to help people make data-driven decisions and predictions [44]. Data analytics involves three types of data retrieval: batch processing of data-at-rest, real-time processing of data-in-motion, and interactive processing in an ad-hoc manner. To analyze a dataset, data analysts often start by exploring the data before they understand 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 get a glimpse of the data through a sequence of exploratory queries before retrieving it for the further analysis [46]. Data exploration, which is an interactive data retrieval process, allows analysts to 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 [43]. Since data exploration is labor-intensive and repetitive, it would be beneficial to analysts if the value-added data derived from the exploration stage could be shared and reused in the future. Data analysts can reuse results from earlier explorations to better streamline the data analytics pipeline. To enable more efficient data sharing and reuse, it is very important to provide provenance information for the data source so that data consumers can be informed about what sort of earlier manipulations have been performed on the data.

DaaS (Data as a Service) enables data provision as a cloud service and provides controlled access to the data through Web service APIs that adhere to the REST architectural constraints (also known as RESTful API) [82]. Some vendors, such as Microsoft Azure\(^1\) and Google Cloud\(^2\), host data stores in the cloud environment and offer data services to facilitate the ingestion, integration, analysis and publication of the data via predefined common sets of RESTful APIs. In order to

\(^1\)https://azure.microsoft.com
\(^2\)https://cloud.google.com/
advocate a standardized approach, OData [1] defines an application-level protocol for interacting with data via RESTful APIs. By following OData, data producers can define their data using the OData data model and publish it using URLs. Data consumers can then access and update the data using simple HTTP messages.

Fig. 1.1 outlines the role of data services in the process of data analytics. After the business problem is identified, it is common practice to use data services to extract a large amount of data into desktop software like RStudio\(^3\) or Jupyter\(^4\), and then to explore this data, followed by the building of analysis models and then by decision-making. These steps can take a couple of iterations or might require a data scientist to obtain other data sources in a different way.

### 1.2 Problem Statements

Data services provide uniform, scalable, and filtered interfaces (e.g. RESTful APIs) for use by data analysts to retrieve data [14]. Many companies and platforms, in-

\(^3\)https://www.rstudio.com/
\(^4\)http://jupyter.org/
including Twitter, Google, and CKAN\(^3\) offer data service APIs that provide simple and easy-to-use access to their data. Data services allow third parties to easily integrate the data resources into their applications. However, these conventional interfaces fall short on data retrieval for data analytics, especially in current practice, as shown in Fig. 1.1.

Firstly, the existing data service design is data-centric but not customer-driven. The current API representations are designed to answer questions according to the underlying database schema and pre-assembled index but lack the specification of analytics domain semantics. As a result, they are only suitable for basic data querying, manipulation, and management [82], but not suitable for responsive, interactive, and comprehensive data retrieval for analytics [26].

Secondly, the current API representations for data services only support one-off queries, which are isolated, static and of no analytics semantics [79]. This communication pattern fails to provide guidance for an analysis process. As a result, data analysts have to blindly request the services many times if they are to understand the features of the data, review the results, and adjust the subsequent queries in order to balance the data coverage and the size of returned data. Intensive labor efforts are involved in data exploration as distinct from data analysis. To acquire data more efficiently, in some cases, data analysts may need guidance from a domain expert.

Thirdly, existing data services provide no standard mechanism for providing context information about the origin, scope, and usage of the data. Although RESTful web services provide self-descriptive information about web resources, which enables their automatic processing [39], the metadata is sent in the header of HTTP messages and restricted to information about the syntax used in the

\(^3\)https://ckan.org/
resource representation. In addition, the semantics of the lineage of the data are less considered, meaning that data analysts remain uninformed about what data exists or how the data is derived and used. As a result, they cannot infer whether these processed data can be reused or not.

1.3 Motivating example

To explore the potential of the 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.

Figure 1.2: Data Exploration Scenario Using Data Service

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.

1.4 Requirements

From this scenario, four main requirements for building data services to explore and retrieve data can be derived:

1. *Clean and consistent representation of underlying data.* Execution of analysis operations depends on the underlying data, which may be available in various formats from various sources. Data exploration process itself is agnostic to these sources and formats. Data services should provide a standardized interface to these sources.

2. *Resource API grounded on data operations.* Data analysis processes invoke operations on the underlying data. Interaction with data hosted on the DEaaS should be similar to that of the downloaded dataset. An analyst should be able to interact with the data services in the same manner as client side data analysis tools. That enables the analyst be able to re-use their data
exploration code on downloaded data by simply replacing API calls with function calls. In other words, resources should be designed in the same way as interfaces on client side data analysis libraries.

3. *Process agnostic resource navigation*. Data exploration, by definition, is unstructured. It is a way of creative learning and experimenting on the available data before analysis. 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. Resource and parameter recommendation approaches should not be based on a dogmatic approach of interacting with data in this phase.

4. *Resource enable data exploration process reuse* 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.

### 1.5 Research Questions

The main research question of this thesis is:

*MainQ. How to design RESTful data services to make the data retrieval efficient and effective for data analytics?*

The main question can be divided into several sub-questions, for each subsidiary research question, detailed technical questions are derived:

*SubQ1. How to model the data retrieval patterns of data analytics?*

- *How to generalize and classify the data retrieval patterns of data analytics?*
Chapter 1

- How to specify parameters and media type of RESTful APIs based on different retrieval types?

SubQ2. How to design the service architecture to satisfy the analytics activity focused data retrieval?

- How to define a clean and consistent representation of underlying heterogeneous data sources to make data exploration process agnostic to any sources with various formats.

- How to design data operations as RESTful resources to support an effective data exploration and preprocessing?

SubQ3. How to navigate resource APIs to assist analysts in the data exploration process?

- How to tailor and apply service discovery algorithm for recommending data query so as to reduce time spent in exploratory analysis?

- How to bring analytics semantics into resource APIs to facilitate the human-in-the-loop processing needed for interactive data exploration?

SubQ4. How to leverage data package techniques to provide data context information, so that data analysts are able to share and reuse the results of the explorative analysis?

- How to specify the data package structure?

- How to embed primitive operations into the data packages?

- How to support heterogeneous runtime environment?
1.6 Research Contribution

In this thesis, Data Exploration as a Service (DEaaS) is proposed to extend the current DaaS for analytics purposes. As shown in Fig. 1.1, a RESTful data service design is provided to facilitate interactive data exploration within a human-in-the-loop process. The computation is pushed onto the data side by providing basic analytics capabilities and lightweight data features rather than by exposing the original big data simply and arbitrarily. Through an interactive conversation between a user and data services, DEaaS continuously recommends and presents resources to the user. In each instance of communication, the user can choose one of the recommended resources as the next step in their exploration. Three semantics of data operations are defined namely drill down, roll up and pan. Through the use of these semantics and historical resource API calls, DEaaS can infer the data space that the user may be interested in and present the corresponding API parameters for the next query. By iteratively interacting with DEaaS, analysts can identify and retrieve the targeted variables and subsets of the data effectively and efficiently.

For each subsidiary research question, detailed technical questions are derived to approach the solution, and the corresponding contributions are listed below:

SubQ1. How to model the data retrieval patterns of data analytics?

– How to generalize and classify the data retrieval patterns of data analytics?

– How to define parameters and media type of RESTful APIs based on different retrieval types?

Contribution: A set of RESTful conversation models for depicting three patterns of data retrieval between data consumers and resources, along with a negotiation protocol to reconcile the data analysts requirements with data providers privacy policies.
SubQ2. How to design the service architecture to satisfy the analytics activity focused data retrieval?

– How to define a clean and consistent representation of underlying heterogeneous data sources to make data exploration process agnostic to these sources and formats.

– How to design data operations as RESTful resources to support an effective data exploration and preprocessing?

Contribution: A RESTful DEaaS data service architecture with an abstract data model and formalized ad-hoc resource design for data retrieval for data analytics.

SubQ3. How to navigate resource APIs to assist data exploration process?

– How to tailor and apply service discovery algorithm for recommending data query so as to reduce time spent in exploratory analysis?

– How to bring analytics semantics into resource APIs to facilitate the human-in-the-loop processing needed for interactive data exploration?

Contribution: A navigation model to help discover and generate data service APIs dynamically. Following the HATEOAS constraints of REST, a discovery tree constructed from the history queries is proposed in order to match the user trace and discover the appropriate resource APIs. The API parameters are recommended according to the different semantics of data operations.

SubQ4. How to leverage data package techniques to provide data context information so that data analysts are able to share and reuse the result of the explorative analysis?

– How to specify the data package structure?

– How to embed primitive operations into the data packages?
– How to support heterogeneous runtime environment?

**Contribution:** A mechanism that extends the *data package* to include data processing scripts and data context information associated with the data, enabling users to customize and reuse the data exploration process and derived data.

Two case studies and one experiment were conducted to enable a qualitatively and quantitatively comparative evaluation of the proposed data service design against conventional data APIs. The results of this evaluation shows that the proposed approach has advantages over conventional data service APIs in terms of REST maturity, interoperability, discoverability, reusability, usability and adaptability. Moreover, the experimental results demonstrate that the proposed resource navigation approach can make DEaaS outperform existing data services in terms of data exploration.

### 1.7 Thesis Outline

This thesis is structured according to the methodology it follows. Firstly, the following **Chapter 2** outlines the background of this research and discusses the problems in existing data service design as regards support for data retrieval for data analytics. **Chapter 3** puts forward the research questions that were developed based on the literature review and discusses the research methodology used to answer these questions and evaluate the research. The proposed data service design for data analytics is discussed in **Chapter 4**. **Chapter 5** evaluates the proposed solution by means of two case studies and one experiment. Finally, **Chapter 6** concludes the thesis and discusses possible avenues for future work.
Chapter 2

Background and Related Work

This chapter identifies the research scope and discovers the research gap by introducing the background and a comprehensive literature review on related work. Section 2.1 first discusses the state-of-art and state-of-practice of data preparation for data analytics. Then, Section 2.2 introduces the data service and its role for data retrieval for analytics. The requirements of data services for supporting data retrieval patterns are discussed as well. Finally, Section 2.3 concludes by identifying the challenges of existing data services when applied to data analytics.

2.1 Data Preparation for Data Analytics

The goal of this research is to fill the gap of existing data service for data preparation for data analytics. Bearing in mind the research target of this work, we investigate the state-of-art data preparation for the analytics. The framework of technology and research works involved in this chapter is shown in Fig. 2.1
2.1.1 Data Analytics

Data analytics is the activity of collecting, parsing and analyzing data to discover patterns and features of data to help people make data-driven decisions and predictions [44]. The lifecycle of data analytics can be divided into five distinct stages that depend on each other, as shown in Fig. 2.2, including problem identification, data retrieval, data preparation, data analysis and result interpretation or presentation. The cycle is a continuous loop, where each stage is iteratively refined based on the analysis result of every iteration. Each of the stages is explained below.

- **Problem identification.** Before a data scientist has a clear purpose of the analysis, he or she needs to have a basic understanding of the domain prob-
Data Retrieval. After getting the domain knowledge of the data, the data scientist needs to acquire the raw data from a broad array of data sources manually or by instruments. Not all of the data are relevant to the identified problem, so the data scientist need to extract data using a proper filter to eliminate irrelevant data. As the data formats could be structured or unstructured, and the access channels to data sources could be static or dynamic, multiple methods are needed to satisfy different data retrieving patterns. More details are in Section 2.1.2.

- Data preparation.

“Scientists spend a lot of time preparing their data for computational analyses. Most of this time is spent cleaning, transforming,
and validating the datasets. When cleaning the datasets, scientists utilize statistics to filter their data, identify and fix outliers or missing data, and rationalize irregularities in data formats (e.g., ad hoc spread sheets). The transformations of datasets consist of changes in format, data type, units of measure, although validation can occur at the level of data and/or schema.” [13]

Data preparation is a critical and time-consuming process for data scientists. It is, however, often an underestimated step in the data analysis process. There are two steps involved in this phase including data exploration and preprocessing. The data exploration involves a preliminary analysis of big and complex datasets, which helps scientists to better understand the characteristics of data they will work with in the following process. Once data scientists know more about the data through exploratory analysis, the next step is preprocessing data for analysis. While data collected might not be in a format ready for analysis, preprocessing operations should be applied. Preprocessing operations include cleaning data, sub-setting or filtering data, transforming data that can be read and understood by programs, such as modeling raw data into a more structured data model or packaging it using a specific data format. If there are multiple datasets involved, this step also includes the integration of multiple data sources or streams.

- **Data analysis.** Given the prepared data, multi-disciplinary techniques and algorithms across machine-learning, statistics and data-mining are adopted during this phase to derive results solving the identified problem. This step can take a couple of iterations on its own or might require data scientists to go back to the start of the process to acquire more data or package new data in a different way.
• **Interpretation/Presentation.** The insights gained from data analysis needs to be reported to decision-makers in the form of visualization, natural language output and so on. This is a very important step to communicate analysis insights and make a case for what actions should follow.

### 2.1.2 Data Retrieval

Data analytics involves three types of data retrieval including batch processing of data-at-rest, real-time processing of data-in-motion, and interactive processing of ad-hoc manner.

In a batch data retrieval, high volumes of data are collected, ingested and then processed to do some relatively time consuming data analytics task. Hadoop MapReduce\(^1\) is an example framework for processing data in batches.

In a streaming scenario, a query keeps getting continuously executed over a stream of data. Thus, rather than batch processing, this analysis allows data scientists to analyze the data-in-motion. This analysis is done in real-time thereby allowing scientists to trigger important events or execute continuous query on data, which returns continuously updated results.

In an interactive data retrieval, a data scientist issues a query, receives a response, formulates the next query based on the response, and repeats. This interactive analysis allow data scientists normally to start by big-picture questions and then continually refine their questions based on feedback and domain knowledge.

### 2.1.3 Data Exploration

As discussed above, data are explored and preprocessed before data analysis. \textit{It is inefficient to pull the whole data into the client side to run the heavy computation,}

\(^1\text{http://hadoop.apache.org/}\)
because this incurs unnecessary data transfer cost and the data may not fit in RAM for a data mining software. Instead, it is desirable to allow analysts to have a glimpse of this data through a sequence of exploratory queries before the retrieval for further analysis [46]. Typically, the analysts start by asking broad questions to get an overview of the data and then iteratively interact with the data system to refine their queries based on the answers and domain knowledge till the final retrieval for further analysis [43]. Data exploration is fundamentally a multi-step, non-linear and interactive process with imprecise end-goals [37].

Data exploration involves a preliminary analysis of big and complex datasets to help analysts better understand the characteristics of data in the following processing and analysis. Data exploration is not a single query but a process of exploratory data analysis (EDA) with multiple steps. EDA [68] employs a variety of summary statistics and visualization techniques, including graphing general trend, sampling important variables, detecting outliers and anomalies, and capturing summary statistics (e.g. mean, median).

Numerous research efforts have been made in the emerging area of database systems for data exploration [43]. A query recommendation system for interactive database exploration is proposed in [20]. In terms of interfaces between users and database systems, many researches focus on assisting users to navigate the underlying data structures by providing visualization tools [11], recommending data objects based on user interests [19, 27, 29], and data sampling [27] to facilitate the data exploration process. However, these solutions offered by different database systems are proprietary in nature, without a unified and declarative exploration interface to present navigational idioms across different systems. This also brings another challenge to leverage the historical user interactions to identify exploration patterns and predict queries [43].
A data warehouse is constructed by integrating data from multiple heterogeneous sources. Data warehousing is the process of constructing and using a data warehouse, which supports analytical reporting, structured queries and decision making. Online Analytical Processing (OLAP) is an approach for supporting analytical queries on multidimensional data models. Data Warehousing and OLAP have been essential elements for data analytics and business intelligence in many industries [21]. Big data stored in heterogeneous data sources (e.g. legacy systems, scientific data repositories, and social networks) can be conveyed and integrated into a structured, well-interpretable multidimensional data model in data warehouse by using batch processing techniques like Extract-Transform-Load (ETL) [23], which is a process in data warehousing responsible for pulling data out of the source systems, transforming, and placing it into a data warehouse. It is demanding and challenging to provide an interface that supports OLAP operations to explore the multidimensional data for analytics purpose [15]. An approach is proposed in [3] to recommend individual or user group OLAP preferences by mining query logs. A recommender system on OLAP cubes is discussed in [33], which leverages former users' investigations to enhance discovery-driven analysis. An RDF data cube is designed to identify and analyze relationships between disparate multidimensional data sources to enable traditional business analytics at the web scale [53]. It is inspiring and helpful to combine the semantics of OLAP data operations with the service resources navigation while exploring big structured data using data service APIs.

Since data exploration is labor-intensive, it would be beneficial for analysts if the value-added data derived from the exploration stage could be shared and reused in analytics community. 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 provenance records the origin and history of the data processing. How
to define and capture the provenance information of the data for the purpose of
reproduction of the data analytics is discussed by providing variety of modeling
tools or extending the metadata [38, 24]. Vistrails [17] is a scientific workflow and
provenance management system that shares some of the same goals, with a focus
on scientific reproducibility. However, these solutions are designed for particular
systems, how to integrate the provenance information into the metadata of data
service is less considered.

2.1.4 Data Integration

As discussed in the Section 2.1.1, data integration is a necessary step for data
preprocessing. Data sources are spread in heterogeneous source systems. It is
necessary but challenging to integrate the structured and unstructured data to a
uniform data format for data analysis. Data integration can be performed on dif-
ferent abstraction levels [28]. According to these levels, the applied techniques can
be grouped into manual integration, developing a middleware application, building
a common data storage and facilitating a uniform data access.

Manual Data Integration

Manual data integration is an process to integrate data manually by data analysts,
which is very time-consuming and labor-intensive. This approach needs the data
analysts to be equipped with deep domain knowledge and understanding of data
sources.
Middleware

A middleware approach is to access various databases and return the merged result to the user by using ODBC and Java Database Connectivity (JDBC) database access drivers. However, it is always onerous for this kind of applications to change or increase the source data systems [71].

Common Data Storage

Building a common data storage and migrating all required information into a new database is a traditional method applied in data warehousing techniques. This process is involved in data ETL process [25]. However, this method need big storage capacity and modeling data before migration. The data schema and business logic should be predefined and well-structured, otherwise, strong effort will be invested into the data transformation, which maps data types and structures from the source systems into data types and structure of the target system. If business environment and data metrics change, the migration process has to be changed subsequently, which becomes very costly and time-consuming.

Uniform Data Access

A uniform data interface provides a logical integration of data and allows users to access data sourced from different data systems. The required information will not be migrated to a target database, but queried from the source databases on demand. In this case, the dynamic data schema matching and reconciling has become a challenging research issue because of the semantic heterogeneity of source systems [71].

Atzeni offered a uniform interface called Save Our Systems (SOS) [4], that allows querying of different NoSQL systems (HBase, Redis, and MongoDB) using a
common set of simple atomic operations. The system proposes a common interface, which implements uniform data retrieval and manipulation methods, and a common data model, which translates and maps the specific structures of the NoSQL systems. However, this solution is limited to specific NoSQL database systems and lacks a theoretical and scalable support for other emerging data systems.

ODBAPI [65] defined a generic resource model to represent the different elements of heterogeneous data stores in a cloud environment, as well as a uniformed REST API that execute the CRUD operations on relational and NoSQL data stores. However, it does not support complex data queries and functions executed on joined data results.

2.1.5 State of the Practice

![Data Analytics Stack](image)

In this section, the characteristics of current big data storage, processing frameworks and tools supporting ETL, analytics and machine learning workloads are investigated in the state of the industry practice.

The objective is to understand the role of our research focus, using a pragmatic data analytics stack as shown in Fig. 2.3. This stack provides a high-level abstract
view combining the traditional enterprise data solutions with big data environment, which comprises four layers, including data resources, acquisition, processing, and application layers.

Basically, data can be ingested from different data sources through data acquisition layers, and then be prepared through data processing layer like cleaning and data model transformation. After that, the well-grained data could be used in application and advanced analytics.

In the data resource layer, data can be structured, semi-structured or unstructured and stored in different data storage systems. Traditional Relational Database Management System (RDMS) stores data in structured data format and supports SQL queries, e.g. MySQL and SQLServer. NoSQL databases are growing in popularity for next-generation Web applications due to its features of high reliability, high performance, high scalability, and real-time read and write. NoSQL systems can be classified into a few main categories [18], including key-value stores (Redis), document stores (MongoDB), and extensible record stores (HBase). Data also can be stored in raw format in file systems (HDFS, S3) and later be transformed to corresponding data frames by data processing frameworks.

Most of above data stores provide query language engines and proprietary APIs for applications to query data. SQL is a typical querying language for relational databases whereas other NoSQL databases (e.g. Neo4j) provide a graph query language (e.g. Cypher) and Hive provides SQL-like queries (e.g. HiveQL) for applications to manipulate and access data. In many of the NoSQL systems, client access is built on RESTful APIs to fetch and store data. Some Java, Python or Thrift client APIs have also been supported. These APIs support CRUD operations or complex queries execution. Nevertheless, there is a gap to fill in terms of developer support. Indeed, each type of data stores exposes different traits, drivers,
APIs, and data models. In most of the time, application developers are lost in this plethora of data stores and they have to manage that by any necessary means. All that will degrade the developer productivity [64]. Some research effort has been done in this area, which will be discussed in Section 2.2.2.

Data processing framework can ingest data directly. In Hadoop ecosystem, many open source processing frameworks access data through data frame abstraction to process data. The processing frameworks mainly contain the batch processing and real time processing. Typically, a processing framework comprises a programming model and an execution engine that is responsible for lower-level resource management tasks, such as scheduling, optimizations and others.

In the application layer, the purpose is to support advanced analytics and custom data applications for extract-transform-load (ETL), business intelligence for data exploration and reporting (e. g. QlikView, Tableau) and advanced analytics tasks, such as data mining and machine learning. The majority of applications access the data resources via query language or the customer API provided by data processing frameworks. Some tools mix the data acquisition and data processing together, which adopts a ELT method to query data in a schema-on-read. So they can apply cleaning and transformation processing on data while extracting it. For examples, some BI applications like Cognos\(^2\) could access the database directly, while others like SAS\(^3\) could approach data source via SQL on the web API. Another example is Spark\(^4\). Data can be ingested from different sources like Kafka, Flume and can be processed using complex algorithms and machine learning algorithms on data streams. For advanced analytics applications, direct access via specific framework API are available, alternatively, they provide customer APIs for

\(^3\)http://www.sas.com/
\(^4\)http://spark.apache.org/
application to process data.

2.2 Data Service for Data Analytics

A literature survey has been carried out to identify the characteristics of data service as well as the gap of existing data service when applied to data analytics. Also, some related work that could be leveraged to fill the gap for the requirement of data analytics are also studied and summarized in this section.

2.2.1 REST and HATEOAS

REST (REpresentational State Transfer) is an architectural style for distributed hypermedia systems and was first proposed by Roy Fielding in 2000 in his dissertation [31]. It defines a set of additional architectural constraints on top of basic client/server architectural style featured by its uniform interface, statelessness of interactions and URI representation. Following REST design principles, a web service is identified by a URI as a resource. Client applications interact with web services through using request-response messages. A RESTful service is not an isolated endpoint, but represents a net of interconnected resources, relationships between resources can be expressed by hyperlinks embedded in representation which refer clients to related resources.

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture that keeps the RESTful style architecture unique from most of other network application architectures. The term “hypermedia” refers to any medium of information that contains links to other forms of media such as images, movies, and text. The principle of HATEOAS defines that the service should embed links in its responses so that the web clients can dynamically
navigate to the appropriate resource by traversing the hypermedia links [51].

However, there is a semantic gap for clients to be navigated by hyperlink in different domain context. To remedy this design flaw in HATEOAS implementation, some hypermedia specifications such as AtomPub and OpenSearch are tailored to achieve the specific application goals. However, the semantics of these domain-specific media types are implicit and generic [60]. AtomPub is designed to cover all the collection-based APIs but cannot reflect different application semantics. Microsoft’s Open Data Protocol (OData) 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 resources focus on data instead of analytics operations, and the related metadata services only present limited description documents [58]. So these hypermedia specifications are deficient to provide semantic guidance for data exploration and analytics.

2.2.2 Data as a Service

Data as a Service (DaaS) enables data provision as web service, and provides controlled access to the data through Web service APIs [82]. Data service is a data-centric web service whose interactions is determined by the underlying data source, which is characterized by its re-usability, flexibility, customization and integration with other applications. In the data analysis process, data services can be regarded as a method for retrieving data. In order to focus on data rather than functional aspects of interactions, the data service bypass the application layer to expose a uniform data interface. Data can centrally reside, be cleaned and enriched as needed, and be exposed or provided to heterogeneous applications, systems or

--5https://bitworking.org/projects/atom/rfc5023.html
6http://www.opensearch.org/Home.
7http://www.odata.org/
users, regardless of where they were [14].

A lot of previous research has been committed on the data-centric web service regarding data model [69] and data operation. In order to define the underlying data model in the data service, [62] proposes an approach to modeling data for data-centric web service based on formal methods and establish a contracting framework as well. Its main contribution is to provide a machine-client readable specification of the data service. In [69], the framework is integrated from both functional and data perspective and based on Business Artifacts. Its contribution is to define a set of customized data retrieval web service operations.

In the industry, more and more websites and platforms like Twitter\(^8\), Google\(^9\) and CKan\(^10\) tend to open their data through RESTful APIs that provide a simple, easy-to-use access to their resources. This allows third-parties to integrate these data services into their own service-oriented applications. In order to advocate a standard way, OData\(^11\) defines an application-level protocol for interacting with data via RESTful APIs. Following OData, data producers can define their data resources in the OData data model and publish data using URLs. Data consumers can access and update the data using simple HTTP messages. Protocols and Structures for Inference (PSI)\(^12\) 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 resource does not define the relationship between these relations. Data analysts do not know how to discover a related data service for data exploration and have to request data services blindly without any guidance.

\(^8\)https://developer.twitter.com/
\(^9\)https://developers.google.com/analytics/
\(^10\)https://ckan.org/
\(^11\)http://www.odata.org
\(^12\)http://psi.cecs.anu.edu.au/.
Database as a Service

In the cloud computing environment, database as a service has emerged as a new paradigm. It hosts data stores in the cloud environment and offers data services to access the data stores via predefined common sets of RESTful APIs [82]. Some products like Amazon SimpleDB\(^{13}\), Microsoft SQL Azure\(^{14}\), Amazon Relational Database Service (RDS)\(^{15}\), MongoDB\(^{16}\) and Apache CouchDB\(^{17}\) provide database service solutions to enable users to store and query data items through web services.

In order to satisfy different business requirements, cloud applications usually need to access and interact with different and non-relational data stores offering heterogeneous types of APIs. Hence, these heterogeneities make users access the unfamiliar data API and migrate data between different data stores. In order to solve this problem, some effort in research and leading database companies has been made currently.

A common data source model defining the different elements represented in heterogeneous data store is proposed in [64]. Meanwhile, they provide a uniform RESTful API named ODBAPI to execute CURD (create, update, query, delete) operations across various relational and non-relational data stores.

A unified RESTful API through which different backend systems and data stores can expose their details is proposed in [32]. They propose a modular ORESTES middleware approach to NoSQL data stores by an scalable tier of HTTP servers.

In [34], a conceptual model for a unified RESTful API for DaasS systems is presented, which aimed to be suitable for all DaasS systems with unstructured data.

\(^{13}\)http://aws.amazon.com/simpledb/
\(^{14}\)http://www.microsoft.com/azure/sql.mspx
\(^{15}\)http://aws.amazon.com/rds/
\(^{16}\)http://www.mongodb.org/
\(^{17}\)http://couchdb.apache.org/
storage or relational systems. This paper aims to offer a base for discussion of the
general notion of a universal DaaS API, with respect to REST API complexity and
performance applying to data as a service.

HTSQL\footnote{http://htsql.org/} enables accessing SQL Server via HTTP arbitrarily, which is an ad-
vanced query language on the web designed for data analysts who have complex
inquiries across relational databases. It wraps the underlying database and repre-
sent the queries by URLs.

**Linked Data as a Service**

Data service have also been studied in the semantic web area, where the data-
providing services and resources are represented in Resource Description Frame-
work (RDF\footnote{https://www.w3.org/RDF/}) views and query-able SPARQL endpoint respectively. SPARQL is
a kind of semantic query language defined by W3C allowing to query very precise
triples selectively in RDF datasets.

To build a domain application protocol over HTTP, which is domain agnostic in
the web application, additional explicit semantics are needed \cite{73}. While RESTful
implementations encode semantics by annotating hypermedia with link relations,
semantics are described by ontologies written in RDF schema and Web Ontology
Language (OWL) in the semantic web area \cite{56}.

In order to fill the gap between hosting data online and publishing linked Data
via a queryable API such as SPARQL, \cite{59} provides an architecture that operates
the semantic web query on an a large scale of heterogeneous datasets.

The Linked Data Fragments (LDF) concept is declared in \cite{72}, which acts as
a uniform access on all Web service APIs to linked data source, consisting of data
entities and controls (hyperlinks). This paper introduces a client-side SPARQL
query processing algorithm to reduce servers payloads. The relation between low availability of public SPARQL endpoints and SPARQL protocol is discussed to find a balance between the query flexibility and the load of server.

### 2.2.3 DaaS for Data Retrieval Patterns

The existing DaaS data services are designed from the perspective of data publisher, using automated tools to generate API quickly and directly from database, focused on the underlying data they have available, rather than the demand of data consumer. Indeed, Since the operation is based on the data, the API is not fully suitable for data analytics in supporting data retrieval patterns discussed in Section 2.1.2.

#### Batch Data Retrieval

In order to analyze data, a big volume of data should be prerequisite in context of data analytics, however, due to the limitation of http cache and payload of the server, existing data service APIs, like the ones owned by Twitter\(^{20}\), Facebook\(^{21}\), and LinkedIn\(^{22}\), do not support whole bulk data retrieval but adopt filters like rate limit and pagination to limit the the scope of data or provide subset of the whole data. Specifically, some leading APIs about how to handle partial responses and pagination are below:

As shown in Fig. 2.4, LinkedIn API queries a person with *ID, first name, last name, and industry*. Google and Facebook have a similar approach, using an optional parameter called *fields* to query the required records. In terms of pagination shown in Fig. 2.5, LinkedIn use *start&count* while Facebook use *offset&limit*. Twit-

\(^{20}\)https://dev.twitter.com/rest/public

\(^{21}\)https://developers.facebook.com/

\(^{22}\)https://developer.linkedin.com/docs/rest-api
ter use pages and rpp (records per page). In this case, to acquire all the related data, a script or tool should be provided to go through all the “pages” of results at a time, then combined to get final result. That means data consumers have to make a huge number of requests to achieve this volume since there is an upper bound for the per page parameter of the API. However, some API set restriction of rate limit to restrict the request times in per time window.

Some API designs use hypermedia or links to show the next data view page. In [36], the collections conversation type is mainly used into acquire large collections by retrieving the target entry and acquire the resources within the collection, the partial representation of which will embed hyperlinks to the first, last as well as the next/previous sets.

However, this method is not suitable for retrieval of subset of batch data selectively. The real requirement of different data size and payload of client is not taken
into account in the web API design. Also, there is no negotiation about batch size and transfer pattern between data publisher and data consumer.

**Interactive Data Retrieval**

A RESTful web service is not an isolated endpoint, but a net of interconnected resources, with an underlying hypermedia model that link not only the related resources but also the possible net of resource state transitions [2].

The conversations used in RESTful architecture are discussed in [36], which presents four widely used types of conversation and summarize their specific characteristics. Based on this, a conversation based design model for REST is proposed by extending the interaction-centric metamodel (ICM) to the conversation-centric metamodel (CCM). By adoption of this model, the design approaches of a RESTful API become much simpler, then the designer of a RESTful API can focus on higher-level functionality provided by a RESTful API instead of trivial details of lower layer.

In order to design an interactive query in the context of data pre-processing workflow, service composition which is the process that combines component services into new service can be a method considered. Some referred studies have been conducted in this area. A set of basic elements of control-flow patterns in the context of compositions of RESTful web services are discussed in [12]. ReLL (Resource linking language), which acts as a hypermedia description language of RESTful resources, makes the application domain semantics both machine and human understandable. The research of Hypermedia-driven service composition [2] is based on Rell and takes advantage of Petri Nets as a mechanism for describing the machine-client navigation.

In summary, the existing data services mostly rely on one-off or request-response
message exchange pattern without negotiation and multiple interactive conversa-
tion consideration, not leveraging Hypermedia from REST principle. Moreover, Hyperlink used in REST service composition is predefined in design time but not adaptive and automotive in run time.

Stream Data Retrieval

In terms of stream data API, a number of technology is studied. WebSocket, which is a protocol providing full-duplex transition channels over a single TCP connection is popular used in stream data API implementation. RESTful APIs support SQL-like query on tabular or column oriented data but do not support streaming request which generate the real time data could be used for complex machine learning algorithms.

A real time data analytics as services architecture [76] uses RESTful web services to wrap and integrate real time data source as data service. However, in this architecture, the real time updating data is not constructed as a relation resource, so it does not realize RESTful streaming service actually.

An architecture of web liquid streams for controlling dynamic heterogeneous streaming systems is proposed in [6]. By means of uniform interface and hypermedia, how to monitor, change and adapt the deployment configuration of a streaming topology at runtime is demonstrated.

In summary, the heterogeneity among the current streaming APIs and the complexity in configuration impede users to access the unfamiliar streaming API and integrate the streaming data. Also, the current streaming API is always tightly coupled with the specific analytics application, which will cause the applications rigid and hard to evolve.
2.2.4 Service Discovery and Recommendation

To help end-users find appropriate services in a large number of existing services, web service discovery and recommendation have attracted a lot of attention in recent years. Many QoS-based approaches [45, 80, 81] are aimed to identify optimal Web services from a set of candidates based on the functional requirement and the QoS (Quality of Service) preferences from users. The approach proposed in [51] analyzes the semantics of the context of intended service to provide effective recommendations. A large body of research [75] uses the semantic web technologies to enrich the semantics of service descriptions for service discovery. An algorithm is proposed in [9] to provide customized web service according to the user-specific demands. goDiscovery [30] builds a model named TF-IDF for the service corps and then navigate the K-D tree to search the model to retrieve relevant services. The approaches discussed above recommend services based on the context information, user preference, and QoS. Instead, the underlying data and the semantics of the analytics process is less considered, so it is not optimal way for data service discovery. Furthermore, most of their implementations are based on simple keyword searches on WSDL service instead of RESTful web services which lack service description in nature [48].

2.2.5 Metadata Service

In RESTful architecture, metadata provides self-describing information about web resources, which enables automatic processing of web resources [39]. 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 [38] 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 REST principles.

A data package is a collection of datasets, metadata information and other data files. It provides a kind of data format for convenient delivery, installation and management of dataset in transparently due to the context information included. Data package protocol defines the format of package, providing a simple, web friendly, standardised and extensible way for users to share and manage data set distributively. However, there is no guidance on how to apply data package for exchanging metadata in RESTful web services. In addition, the scripts in the package is not reusable, and the description of metadata can not reflect the domain knowledge, which refers to analytics data schema.

2.2.6 Other Related Work

Data exploration as a service is proposed in [7], which uses data summarization and relevance techniques to explore relevant data iteratively based on multidimensional data models. However, the solution lacks necessary data statistic operations to explore data effectively and enough user involvement to refine the service recommendation iteratively.

In [79], an analytics activity focused API design for data services is proposed. The main resources are designed for retrieving data and a navigational model based on analytics semantics is proposed to explore data. However, the navigational model is based on predefined semantics which is derived from OLAP operations while users’ behavior is not taken into account. To optimize the navigation model to make the recommended resource target the user’s need is worthy to be discussed.

GraphQL is developed by Facebook in 2012 and acts as an alternative to REST.

\footnote{https://specs.frictionlessdata.io/data-package/}
GraphQL is an advanced query language over HTTP and a server-side runtime query engine for fulfilling those queries by using a customer-defined data type system. Compared to OData, it allows for more efficient retrieval of data by enabling users to fetch multiple and nested resources in a single request. However, due to the huge and complex query, it may have unexpected side effects when scale the data volumes and server usage. In terms of functionality, it cannot satisfy rich analytics operations or provide mechanism to enable domain guidance.

### 2.3 Summary and Conclusion

This chapter has provided an overview of activities and characteristics of each stage among the lifecycle of data analytics. The discussion focuses on the data preparation for data analytics by analyzing the data retrieval patterns and data exploration features. The spectrum of tools and technologies available to domain users in the state of the art big data environment are also surveyed.

After the background of data analytics, the concept of “Data-as-a-Service” in the state of art and practice was comprehensively reviewed following a brief introduction of RESTful web service and related technology. There are some gaps in existing data service when applied to data analytics in the following directions to which improvements can be provided:

- Existing one-size-fits-all API design for data services falls short on supporting responsive, interactive, and comprehensive data retrieval for analytics. The conventional CRUD-based 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[26]. Thus, they are only suitable for basic data querying, manipulation, and man-
agement [82], and do not support data analytic operations well. The issue on how to explore the underlying data space is rarely considered in the current DaaS.

- The API representations of existing data services only support one-off queries, which are isolated, static and of no analytics semantics, and thus failing to provide guidance for an analysis process, which involves multiple and iterative steps. As a result, data analysts have to blindly request data services multiple times to understand the features of the data, review the result, and then adjust the subsequent queries to balance the interesting information and the size of returned data. Intensive labor efforts are involved in data exploration rather than data analysis.

- 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. As a result, the data exploration process and the derived data generated from data analysts can not be reused in the data analytics community as discussed in Section 2.1.2.

To fill the gap discussed above, some related work which could be leveraged to fill the gap for the requirement of data analytics are also studied and summarized in this section. As data exploration can be regarded as a process for users to discover the appropriate service to explore data, service discovery and recommendation in the business process were surveyed. To provide context information in service design, the current state of metadata service were investigated. Some latest technology in data exploration services were also reviewed.
Chapter 3

Methodology

Research is an art of scientific investigation, the goal of which is to discover answers to questions through a systematic research process [49]. The research process shares many similarities with having a journey since both of them need to explore the answers for two principal questions. The first is to decide what you want to find out about or, in other words, what research questions you want to find answers to. The second is to decide how to go about finding their answers, after research questions or research problems are decided. The path to finding answers to your research questions constitutes research methodology [50].

This research followed the empirical research methodology, which is evidence-based research, coming up with conclusions which are capable of being verified by observation or experiment [50]. The rigorous case study methodology [77] was combined with simulated and real-life cases. The relevance and significance of the theoretical contribution can be validated by the case study and empirical results. This research adopted both quantitative and qualitative approaches. The quantitative approach involves the generation of data in quantitative form which can be subjected to rigorous quantitative analysis in a formal and rigid fashion. The qual-
itive approach concerns the assessment based on the defined quality attributes.

Specifically, based on the guideline provided in [50], the methodology is customized intentionally for this research as shown in Fig. 3.1

![Figure 3.1: Overview of the research methodology](image)

After a broad literature review and research problem identification, the research questions are formulated and listed in Section 3.1. The methodology used to answer the research questions is given in Section 3.2 which includes two parts: one is literature review on research and evaluation methodology, and another is propose
the solutions to address the research problems and answer research questions. Section 3.3 discusses the methodology adopted to validate and evaluate the solution. During the process of evaluation, the solution is refined iteratively until the evaluation objective is achieved as expected. It is noted that the literature review is continually throughout the whole research.

3.1 Research Question Formulation

Research question formulation is of most importance in the research process [50]. The research problems are found based on the background and related work through an extensive literature review (discussed in Chapter 2). The knowledge in the data analytic is broadly studied, within which this research contributes and is undertaken. After a careful assessment of whether existing data service approaches support data analytics activity-focused data retrieval, the research gap in current design approach of data services is identified in Section 2.3. Based on the data retrieval patterns and data exploration features, this research was conducted within the area of data service architecture and the API design for data retrieval activities. The goal of the research is to provide a RESTful data service design which specifies data retrieval interface supporting responsive, interactive, and comprehensive data retrieval for data analytics. Based on this goal, the main research question is derived in Section 1.5.
Chapter 3

3.2 Methodology for Deriving Research Solution

3.2.1 Literature Review

The literature review was conducted in the following area including data analytics, data exploration, data service design, service discovery and RESTful API design in Chapter 2.

In this chapter, the literature review is conducted at the research methodology level. It makes contribution to all following operational steps, including:

- Selecting an appropriate research methodology that is capable of providing validated answers to the proposed research questions. The research methodology discussed in [50] is selected, customized and applied in our research.

- Defining evaluation objective, method and measurement criteria to evaluate the research solution in a rigorous way. Case study and experiment methodology are both adopted in our research, more details are discussed in Section 3.3.

3.2.2 Research Solution Development

In this section, the theoretical solution is proposed to solve the research problem. The specific solution in this research includes:

- Design a set of RESTful conversation models for the interaction between data consumers and resources.

- Design a REST-based data service architecture and its resource APIs to realize core functions of data retrieval.

- Design a navigational model for analysts to navigate resource APIs on the purpose of data exploration.
• Design a mechanism using extended data package to contain, publish and share data processing scripts and data context information.

Based on the architecture solution, the DEaaS API framework is developed and a set of API library are implemented. The API framework is used to preliminary validate the solution. In return, the result of the validation was further used to enhance the solution. Details of the proposed solution are discussed in Chapter 4.

3.3 Methodology for Evaluation

Empirical methods such as controlled experiments, case studies, surveys and post-mortem analyses are needed to help us evaluate and validate the research results [47]. There are two main approaches to do empirical studies. Qualitative research concerns studying objects based on a range of different ways of interpretations, while quantitative research concerns comparison and statistical analysis based on experiments or collecting data through case studies. These two approaches should be regarded as complementary rather than competitive [74]. In this research, both approaches are adopted to address different questions. Depending on the purpose of the evaluation, we selected case study and experiment methodology to validate our solution quantitatively and qualitatively.

3.3.1 Case Study

Case study is a research method to investigate a single entry or contemporary phenomenon within specific context [61] [77]. The metrics for selecting case study methodology [77] includes the type of question, control over the behavior events and generality of contemporary phenomenon in a real-life context. Specifically, we selected case study methodology to evaluate our research because of several rea-
sons. First, case study methodology was originally used primarily for exploratory purposes, and some researchers still limit case studies for this purpose, as discussed by [42]. This research is an exploratory research to find new method and generate hypothesis in software engineering research, so case study is appropriate for the research purposes. Second, case study creates an explanatory scenario for validating if the architecture can achieve the quality attributes which are a set of measurable or testable properties that is used to indicate how well the system or design satisfies the needs of its stakeholders [22]. Thirdly, the data exploration processes chosen for the case studies in this research are the typical and representative data exploration processes running in the real-world.

Within software engineering, case studies are not only used to evaluate how or why certain phenomena occurs, but also to evaluate the differences between, for example, two design methods [41]. In this research, case study methodology was adopted to compare the proposed approach with the OData service design.

There are five steps in a typical case study process including case design, preparation for case study, data collection, and data analysis [42] [77] [61].

- **Case design** is the first step of case study process, which specifies the likely cases to be studied and the units of analysis. In case studies, the case should be selected intentionally relevant to the study proposition from either literature or real world [77]. Multiple-case design method [77] was applied for this research on a representative synthetic dataset and a real-life dataset. Case design also involves developing the hypothesis of the case study and the data collection procedure. **Case study plans introduced how to prepare these two cases respectively in Section 5.2.1.**

- **Preparing for case study** mainly focuses on the case study protocol development, which serves as a guidance for collecting case study evidence in the
following process. Case study protocol is developed by the investigator and reviewed by the stakeholders and other researchers. In this phase, the formal approval for the plan to protect the human subjects in the case study was obtained. UNSW Human Research Ethics Committee has formally reviewed this research project and granted the ethical approval.

- **Data collection** follows the procedure proposed in the design stage to collect data from different data sources. In this research, data are collected from direct observation and artefacts of design and implementation.

- **Data analysis** is the last step, which draws empirically based conclusion to answer the research questions using quantitative or qualitative analytic techniques. In our research, the qualitative data analysis is based on quality attributes, such as interoperability and discoverability, while the quantitative data analysis is based on the performance metrics.

The details of the case study are discussed in Section 5.2 and Section 5.2.4

### 3.3.2 Experiment Methodology

Experiments are often conducted to compare a number of different techniques, methods, working procedures, etc. For this type of studies, methods for statistical inference are applied with the purpose of showing with statistical significance that one method is better than the others [74].

The experiment methodology of an empirical research defines several guidelines in below areas [74] [47]:

- **Experimental context** describes the background information in which an empirical study is conducted or the technique is developed. It also discuss
the research hypothesis and the related work to ensure the objective of the research is properly defined.

- **Experimental design** describes the products, resources and processes involved in the experiment. In this stage, it is necessary to define the measurement criteria which are relevant to the objectives of the study to confirm that the design is appropriate to meet the study objectives.

- **Conduct of the experiment and data collection** involves committing experimental participants, preparing instrumentation, executing and collecting the experimental data. As the measures are not standardized in software experiments, the data collection process must be defined well enough for the experiment to be replicated.

- **Analysis and Interpretation** applies descriptive statistics to analyze the collected data and present quantitative results as well as significance levels in graphical or tabular representation. The interpretation or conclusion should be provided following the results.

This research evaluation is aligned with the above discussed guidelines of experimental methodology. The experiment environment and process were designed to compare the proposed approach with the service design which is an industrial standard for building data service APIs. The performance metrics were defined based on the purpose of evaluation. We used one synthetic and one real-life dataset to execute the experiments. Afterwards, the experiment results are collected, validated and analyzed. The discussion of threats to validity is also included to specify the possible limitation of the study. More details about our experiment can be found in Section 5.3.
3.3.3 Conclusion

This research adopts both case study and experiment methodology to evaluate the proposed solution. The difference between case studies and experiments is that experiments sample over the variables that are being manipulated, while case studies sample from the variables representing the typical situation [41]. These two methods can be complementary in that they can be applied at different stages in the research process. Following the empirical research methodology, our research combines the advantages of case study and experiment methods to answer the different research questions qualitatively and quantitatively.
Chapter 4

Data Service Design for Data Analytics

In this chapter, corresponding to the research problems summarized in Section 2.3, a novel approach for designing RESTful data services for data analytics is proposed. A set of RESTful conversation models are introduced to depict the interaction between data services and clients for different data retrieval patterns in the Section 4.1; To fill the gap in current data services for supporting online data exploration, Section 4.2 introduces the concept of Data Exploration as a Service (DEaaS) and its RESTful data service architecture, which describes the data model and key service resource to facilitate the data operations in data analytics. The resources in the service architecture are interconnected to each other according to the context of the data analytics. In order to guide user find the appropriate resource for efficient data exploration and retrieval, a resource navigation model which defines the domain application protocol is designed in the Section 4.3; To assist users to share and reuse the data exploration process, a mechanism using extended data package to contain, publish and share data processing scripts and data
context information is proposed in the Section 4.4; Finally, Section 4.5 concludes the solution.

4.1 Conversation Models

This section discusses the characteristics of conversations in REST architecture and propose to use conversations as a modeling tool to depict the data retrieval patterns for the design of RESTful APIs for data services.

RESTful Conversation

The importance of conversations and the need for services to describe the supported conversation types was originally discussed in the context of messaging middleware [40]. RESTful conversation is a well-known concept in service design to describe interactions between a client and one or multiple resources. The route of a RESTful conversation is controlled by the resources and dynamically discovered by the client involved within the conversation. Whenever a client interacts with a resource, this resource is either the starting point of a conversation or the client has been forwarded to this resource by following the hyperlinks embedded in the representation of another resource.

The relation between conversations and REST architectures is studied in [36], which introduces four widely used types of conversation and summarize their specific characteristics. A conversation based approach for modeling REST is proposed by extending the interaction-centric meta model to conversation-centric meta model. The author collects four practical examples of conversation to demonstrate conversation types by showing the sequence of request/response communication activities listed in a log of the HTTP interactions and also visualized using UML
sequence diagrams as follows: **Redirect** (a client request a resource which then redirect the client to another resource); **Accessing collections of Resources** (how to deal with a container for other resources); **Try-Confirm-Cancel** (the resources that have been temporarily reserved will autonomously revert back to their original state, unless they are confirmed within a given timeframe); **long Running Requests** (the client track the progress of the request by instant polling mechanism). By comparison of existing approaches for creation of RESTful APIs with a model driven approach [35] for REST compliant services whose main idea is to follow a model driven software design (MDSD) approach for the design and realization of REST APIs. This work of [35] is extended and refined in [36] by introducing an interaction centric meta model as atomic resource model as well as a conversation centric meta model as composite resource model. Based on this, modeling approach in terms of conversation instead of modeling single resources and their basic interactions is recommended due to its simplicity. Subsequently, the transformation from conversation-centric model to interaction-centric model is introduced as expansion is introduced to integrate the conversation metal model to model driven software development approach.

By adoption of this conversation-centric approach for modeling REST API, the description and specification of a RESTful Web API can be greatly simplified, then we can focus on higher-level capabilities of a REST API instead of lower level design details.

**Interactive Conversation Model**

In order to reduce the computation cost and improve the timeliness of analysis, the interactive analytics involves the human insight into the process of data analysis. In an interactive retrieval process of data analytics, the data analyst issues a query,
receives a response, formulates the next query based on the response, and repeats the process. The interactive retrieval process allows the analyst to start from general, big-picture questions and then iteratively interact with the dataset to refine their query based on feedback and domain knowledge [37], [43]. As shown in Fig. 4.1, 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.

During the interaction process, requests are checked against the privacy policy of the resource, which is defined by the data provider. The policy constraints are imposed on the different resources and performed by a privacy Compliance Checking Module (CCM). If the query issued by the client is incompatible with the privacy policy, the resource will refuse the request with a notification and navigate the client to other resources with anonymous data or expose the data partially. For example, when the client requests part of customers data that could identify a specific customer and expose her privacy, the resource may respond with an error and guide the client to retrieve the aggregation data. Through this interactive interaction, client and resource could make a negotiation and reach the deal.

**Batch Conversation Model**

Batch retrieval means a group of operations are executed for the dataset in batch. Data are collected to a store to do relatively time-consuming data analytics tasks. For very large datasets, the request is processed asynchronously, and a link pointing to the dataset is returned immediately to the client for status monitoring. The main interactions of batch conversation model are shown in Fig. 4.2, a client makes a request to the resource packager which is responsible for extracting data from the
data source and executing operations on the data. *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.

**Streaming Conversation Model**

In the streaming data retrieval, a client can receive the real-time updates or events occur 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. 4.3, the *client* sends a request to 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.
Figure 4.2: Batch Conversation Model

Figure 4.3: Stream Conversation Model
4.2 Service Architecture

To address the requirements described in Section 1.4, Data Exploration as a Service (DEaaS) is proposed to fill the gap in current DaaS for analytics purposes. DEaaS service architecture is tailored to facilitate interactive data exploration within a human-in-the-loop process. The computation is pushed onto the data side by exposing basic analytics capabilities and lightweight data features, instead of the original big data as web services. Through an interactive conversation between a user and data services, DEaaS continuously recommends and presents interesting resources for the user to explore and retrieve data.

4.2.1 Overview of Service Architecture

The layered DEaaS architecture with the main resources involved in the respective conversation models, as shown in Fig. 4.4, consists of three layers:

- **Data Model** layer, which provides a standard interface to the underlying...
multiple data resources. This base layer of the architecture organizes data in a multidimensional data model to describe the structure of the information exposed by the data services in the upper layer.

- **Resource** layer, which provides different operations for data exploration purpose. These operations are designed in the same way as the interface on current popular data analysis tools, e.g. Spark API library\(^1\). A set of necessary operations are provided only for data exploration on our defined data model layer, which can also be extended based on increasing requirements. These operations are grouped into five categories: **filter**, **aggregator**, **sampler**, **statistic function** and **packager**. Through an interactive conversation between the user and data services, DEaaS continuously recommends and presents interesting sources to the user by presenting links and/or suggesting parameters to make API calls.

- **Resource Navigation** layer, which is responsible for searching the proper resource APIs and related parameters for analysts to explore data efficiently using three components, whose details are discussed in Section 4.3:

  - **Metadata service**, which provides information on the dataset schema and the available resources;

  - **Discovery tree**, which helps users to explore data based on previous data exploration queries. It is achieved by maintaining a specific data tree structure that holds the queries executed in previous sessions;

  - **Parameter recommendation**, which shows recommendations on possible interesting parameters settings that rely on three defined types of data analytics operations: Roll up, drill down, and pan.

\(^1\) [https://spark.apache.org/docs/latest/api.html](https://spark.apache.org/docs/latest/api.html)
4.2.2 Data Model

In this layer, multiple data sources in disparate data formats are integrated into a clean and consistent data model to hide the underlying details. In terms of data model selection, multidimensional data model supports multi-step and iterative exploration process of big data because it provides a representation of data that allows aggregation of data according to different measures and dimensions. These representations are related to the observed problems, thus give proper semantics to the collected data [8]. This case leverages and formalizes a multidimensional data model to describe the structure of the data exposed to the service resources.

- **Relation** is an independent data resource in our architecture. It contains a collection of data instances which are described by a set of properties named **attributes**. (e.g. 2015-2017 Aus. travel survey may contain area_code, total_distance, etc). For simplicity and straightforward presentation, one relation can be assumed as a flat table where attributes represent columns.

- **Dimensions** are categorical attributes of a dataset. Based on the data features, dimensions fall into two categories: atomic and composite. A composite dimension is hierarchical, e.g. year-quarter-month-day for the *time* dimension. An atomic dimension is a unique attribute. e.g. the “Paris” for *city* dimension.

- **Measures** are quantitative attributes that can be measured or aggregated by one or more dimensions like the number of sold properties or household income. Some continuous attributes can be reformatted as discrete based on user’s definition, like *age* that acts as a dimension when the values are categorized into a group, like $20 < age < 40$. 
• **Conditions** are the arguments of the query used to reduce the size of the data returned from the resource. They are based on dimensions and measures. A set of JSON-based built-in query schemas is adopted in DEaaS to define query arguments functions e.g. condition: \( \text{dim} = "x" \text{ AND dim} = "y" \), params: \(["value", 1]\).

In order to provide a unified and consistent representation for the upper service resources, the components of data model introduced above are formalized and presented in Table 5.2 (Rows 1-4). These definitions are the foundation on which we will build the semantics of drill down, pan, and roll up for resource navigation in Section 4.3.

The data model layer normalizes the underlying multiple disparate data resources. It decouples data resources from the service resource and navigation design, which satisfies the first requirement of standard interface to data sources in Section 1.4.

### 4.2.3 Service Resources

Every data resource has a set of data operations and related parameters based on its dimensions and measures. These data operations resemble the function signatures of client-side data analysis libraries. We classify these operations into five categories.

- **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. A filter operation can specify one or more measures on one or more dimensions meeting conditions. Take SQL statement as an example, *select* the measures and dimensions restricted by *where* clause.
Table 4.1: Definition of Resource Model Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D = {d_1, ..., d_k}$</td>
<td>A set of full dimensions on a relation $R$ where $k =</td>
</tr>
<tr>
<td>$M = {m_1, ..., m_n}$</td>
<td>A set of full measures on a relation $R$ where $n =</td>
</tr>
<tr>
<td>$Op(D_o, M_t)$,</td>
<td>A set of operations on $o$ dimensions and $t$ measures, where $o \leq k$ and $t \leq n$</td>
</tr>
<tr>
<td>$D_s \subseteq D_o, M_c \subseteq M_t$</td>
<td>dimensions, where $A$ is the set of possible values for $M_c$ and $B$ is the set of possible values for $D_s$</td>
</tr>
<tr>
<td>$Cond(M_c : A, D_s : B)$,</td>
<td>A condition on $c$ measures and $s$ dimensions, where $A$ is the set of possible values for $M_c$ and $B$ is the set of possible values for $D_s$</td>
</tr>
</tbody>
</table>

- **Aggregator** allows the data analysts to have a summary characteristic of the data without extracting specific 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. It contains a set of summary operations, where the data is grouped into the chosen dimensions, eg. *sum, mean, max* and *min*. The results from the aggregator can be reduced by conditions.

- **Sampler** returns a sample of data from a relation. It supports various sampling algorithms, e.g. random sampling. This allows analysts to quickly build
and test their models within a sample of data that can fit into the client’s memory. It provides sampling operations, like random sampling that samples a specified size of data on various dimensions and measures with or without replacement.

- **Statistic Function** provides a set of statistical operations over the data, which effectively assist data analysts to discover the relation, understand general trend, and outlier of the data as auxiliary means. These operations are executed on the server side, and only the chosen statistic is returned, e.g. `isnull` is a necessary function for data cleansing, which is used to check for the missing value to help analysts estimate the data quality. Then analysts can take several 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. `frequency` is the number or ratio of occurrence of a dimension value in a dataset. `correlation` represents the relationship between two measures. Specifically, given the attribute `gender` and a representative population of people, the frequency of gender `female` occurrence is 50%.

- **Packager** retrieves the big volume 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 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. More details are in Section 4.4.

Formally, to ensure these data operations are straightforward to the underlying data model in a consistent representation, we denote data operation as $Op(D_o, M_t)$,
as shown in Table 5.2 (Rows 5-6), where $D_o$ is the dimension and $M_t$ is the measure. Given a relation $R$, the instances $[I_1, ..., I_n]$ returned from a DEaaS resource $R(Op(D_o, M_t), [Cond(M_c : A, D_s : B), GroupBY(D_g)])$ is the value obtained by selecting or applying a set of chosen measures $M_t$ and dimensions $D_o$ meeting conditions $Cond(M_c : A, D_s : B)$ or summarizing through $GroupBY(D_g)$ optionally.

Provided with resource APIs, analysts can explore the data by manually making API calls or clicking links in the response. Alternatively, they can write client-side scripts that make API calls. After exploration, analysts may decide to download the dataset and then redo some of the operations used in the exploration phase. If they wrote scripts on the client side, they would be able to reuse most of it by switching to function calls instead of API calls. This satisfies the second requirement in Section 1.4.

### 4.3 Resource Navigation

In a RESTful architecture, the HATEOAS principle requires that services respond with hypermedia content that allows triggering state transitions in the client application [31]. DEaaS leverages HATEOAS to navigate users in discovering optimal resources. It can recommend possible next steps for the interactive exploration of large datasets, and provide the information of the relationship between the resources in the context of data analytics. Users could effectively make a data exploration by following one of the links embedded in the representation to the next resource.

To manage the service request and discover the optimal resource, Section 4.3.1 first propose to enrich and leverage HATEOAS with semantics to navigate resource, then Section 4.3.2 introduces an enhanced 3-step approach to improve the efficiency
and effectiveness of the resource navigation.

4.3.1 Semantic Navigation Model

In a RESTful architecture, the HATEOAS principle requires that services respond with hypermedia content that allows triggering state transitions in the client application [31]. Specifically, the \texttt{links} property is used to represent all the actions and resources related to each resource. Users can select one of the \texttt{links} to follow as the next step. \texttt{links} is defined as an array of Linked Description Objects (LDOs) in JSON Hyper-Schema\footnote{http://json-schema.org/latest/json-schema-hypermedia.html}, which obeys HATEOAS principle and assists discovering all the related resource API templates with the current resource API. Each LDO at least contains one \texttt{href} property, which is the target of the link, and a \texttt{rel} property indicating the relationship between the linked resource and the current resource. Users could effectively make a data exploration by following \texttt{links} embedded in the representation to access the next useful resource.

The resources in the DEaaS are interconnected to each other according to the context of the data analytics. DEaaS leverages HATEOAS to navigate users in discovering optimal resources. The semantic navigation model which incorporates the analytics domain semantics into HATEOAS is designed to assist clients to form their queries for interactive exploring large datasets. Based on this model, DEaaS can inform 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. 4.5. The circles represent the resources introduced in the Section 4.2.3, 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.

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 semantic navigation model—parameter values are provided to instantiate the API templates based on the relationship to 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 semantic 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. It can present possible next steps for the interactive exploration of large datasets, and provide the information of the relationship between the resources in the context of data analytics. Users could effectively make a data exploration by following one of the links embedded in the representation to the next resource.

We present their schemas of Links separately in Listing 4.1 and Listing 4.2.

```json
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Schema defining links between resources",
    "type": "array",
    "items": {
```
Listing 4.1: HyperSchema of Links for Dynamical Discovery

```json
{
    "$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 4.2: HyperSchema of Links for Dynamical Generation

As shown in Listing 4.1, 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 4.2 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.

### 4.3.2 Resource Navigation

Through the semantic navigation model, users could make a data exploration by following one of the links embedded in the representation to the next resource in a transparent way. However, the data service only define the semantic and present the relationship between the resources in the context of data analytics. In this section, we will introduce how to recommend the service resource and API parameters to make the interactive data exploration more efficiently and effectively.

To manage the service request and discover the optimal resource, our resource navigation involves three parts. (i) Metadata Service presents all URIs of the resources available at the dataset, (ii) Discovery Tree records the paths of all query
history to recommend the optimal resource to the user, and (iii) Parameter Recommendation offers three parameter selections associated with each resource based on a set of pre-defined analytics specific semantics and the user’s last input.

**Metadata Service**

Metadata Service of a dataset is a self-describing resource that exposes the metadata defining data schema and resources available at this dataset like type of data source (e.g., dynamic, static), data size and description about this dataset, which helps the data users to have an initial understanding about the dataset. The metadata service is presented as a single entry URI, which can be queried via HTTP GET request. Data users interact with the metadata service to discover the resources related to the dataset. The response of metadata service contains a list of URIs for collections of the different groups of resources, for example, filters, samplers, and statistic functions. Then the user can send a HTTP OPTIONS request to acquire assistance on how to invoke a specific API.

**Discovery Tree**

Since data exploration is labor-intensive and repetitive, it would be beneficial for analysts if they could learn from the explorative analysis done by others. Navigating
the most popular exploration process is a sensible choice for analysts, especially for beginners to quickly address the core question. The historical queries from other analysts helps the analyst with similar hypothesis navigate a similar exploration process.

To find the most popular pathway, we accumulate the number of queries of each resource as frequency and normalize it into a transition probability. This approach allows the analyst to be navigated to the resource(s) with high transition probability. The user is able to jump to other resources with lower transition probability through the metadata service of the resource.

We define query session as a set of queries for data exploration from a session of a user. It is stored in the form of a resource invocation sequence, namely trace, which contains the name of the resources and the sequence of their invocation. For example, the sequence $\langle R_1, R_2 \rangle$ represents that the user called resource $R_1$ before calling $R_2$.

We construct a rooted tree named discovery tree to summarize all traces of resource invocation of a dataset. Fig. 4.6(a) shows an example of a discovery tree constructed with two traces: $\langle R_1, R_2, R_3, R_2 \rangle$ and $\langle R_1, R_3, R_2 \rangle$. (Note: $R_0$ is the root of the tree, which represents the gateway of all services). Every trace in a query session forms a path on the tree. Each node is annotated with a resource name and its frequency of resource invocation. Every edge from a node to one of its children is assigned a transition probability generated from the frequency of its children. Transition probability of a node indicates the value of a chance the resource represented by the node can be followed by another resource.

The example in Fig. 4.6(a) shows that both traces start with resource $R_1$, which is always followed by $R_2$ or $R_3$, and the proportion of traces that transit to $R_2$ is same as that of $R_3$. Therefore, the transition probability of the edge from $R_0$ to
$R_1$ is set to 1, and the transition probability of the edge from $R_1$ to $R_2$ and from $R_1$ to $R_3$ set to 0.5 each. When a new query session with the trace $\langle R_1, R_2, R_3, R_4 \rangle$ completes, the tree needs to be updated. As a result, a new node, $R_4$, is added, and the transition probability of the edge from $R_3$ on this branch is updated to 0.5 each.

Algorithm 1 outlines how the discovery tree is built and updated. Given the discovery tree and the sequence of resource invocation, the algorithm recursively finds the matched trace in the tree and updates the corresponding nodes and edges (lines 1-12). If the match fails, a new node is added to the tree (lines 13-17).

During the query session, we can guide the data exploration by searching the discovery tree and using the transition probabilities to recommend which resource to invoke next. We search for an exact match of the sequence of resource invocation by traversing the discovery tree from the root. However, a query session may generate a sequence which cannot be matched in the tree. In this case, we navigate the trace by searching for the longest prefix match between the current sequence and the discovery tree.
Algorithm 1: UpdateTree

Input: $T, S$ //Root of discovery tree, a sequence of resource invocation

Output: Updated $T$

1 Procedure UpdateTree()
2     MatchFlag = False;
3     if $S == \text{Null}$ then
4         return $T$;
5     end
6     for each child in $T$ do
7         if child.name == $S$.name then
8             child.frequency ++;
9             MatchFlag = True;
10            return UpdateTree(T.child, $S$.next);
11         end
12     end
13     if MatchFlag == False then
14         $T = T$.addchild($T$.name);
15         // addchild() function will add the new child and return the added child
16         return UpdateTree($T$, $S$.next);
17     end
18 updateEdge($T$.transitionProbability);

The longest matching prefix is proposed by searching for an exact rooted subtree match using the pre-order list representation of trees in [52]. This prefix match approach was applied in the context of business process simulation with historic logs [63]. Our approach deals with resource invocation instead of logs, and update the tree during runtime.

For example, the entered sequence of resource invocation associated with Fig. 4.6(b) is $S = \langle R_1, R_5, R_2, R_3 \rangle$, while the discovery tree only contains the trace $\langle R_1, R_2, R_3, R_2 \rangle$. When the query session reaches a sequence of $S' = \langle R_1, R_5 \rangle$,
resource $R_5$ and the corresponding transition probability for the last invocation of $R_5$ cannot be found in the discovery tree yet. In this case, for $S$, the longest matched sequence is $\langle R_2, R_3 \rangle$. We call this match the longest prefix match because the match is found by searching prefix of the last resource invocation $R_3$. We find $R_3$ and sample its children according to corresponding edge’s transition probability.

The longest prefix match is found by searching for the list representation of trees. The trace list representation of a tree $T$ is defined as follows: $L = [l_1, l_2, ..., l_i]$ where $l_i$ is a trace of the tree. During each trace traversal, a node is added to $l_i$ until no child nodes exist. For example in Fig. 4.6(b), the tree is presented as $T = \langle \langle R_1, R_2, R_3, R_2 \rangle, \langle R_1, R_2, R_3, R_4 \rangle, \langle R_1, R_3, R_2 \rangle \rangle$. The sequence of resource invocation $S$ will search each list to find the longest prefix match. Algorithm 2 shows how to search the discovery tree using the sequence of resource invocation. The algorithm (lines 1-12) looks for a match from the root of the tree. If it fails, the match will start from the tail of the sequence by partialMatch function, which returns a list of matched trace (lines 17-27). When the match can no longer be found, the algorithm samples a candidate resource from a trace based on the transition probability (line 15).

In the application, the analyst follows one of the links embedded in the response of a resource and submits a query. Our approach traverses the discovery tree with the current trace of the analyst to retrieve the relevant resources with the highest frequency of query. After the analyst completes all the queries and the session
expires, the tree is updated with the query session.

**Algorithm 2: SearchTree**

**Input:** T, S //the root of discovery tree, a sequence of resource invocation

**Output:** R //A recommended resource

1. Procedure `SearchTree()`
2.     MatchFlag = False;
3.     if S == Null then
4.         R = Sample(T); //sample T’s child based on the transition probability
5.     end
6.     for each child in T do
7.         if child.name == S.name then
8.             T ≡ T.child; S ≡ S.next;
9.             MatchFlag = True;
10.            return SearchTree(T, S);
11.         end
12.     end
13.     if MatchFlag == False then
14.         L = partialMatch();
15.         R = SampleTrace(L) // sample a trace and corresponding node based on the transition probability
16.     end

17. Function `partialMatch()`
18.     // L is the trace list of T
19.     for i = 2, i<length(S), i++ do
20.         S' = S[length(S) - i]; // tail of the sequence
21.         for each l_i in L do
22.             match l_i with S';
23.             if match exist then
24.                 L.append(l_i);
25.             end
26.         end
27.     end
Parameter Recommendation

During the interaction with a resource, the user tweaks the parameters of resource API until she is satisfied with the results. In order to facilitate this process and help the analyst approach the targeted data space efficiently and precisely, our data service recommend the parameter values to instantiate the API templates through analyzing the past parameters provided by user. The links embedded in every HTTP response reveal the relationships between the recommended parameters and the previous ones. According to the semantics of the links, the parameters provided by the user are analyzed and new parameters are recommended using the algorithms discussed below. In Section 4.3.1 and Section 4.2.2, we defined and formalized three data operations on our data model: Roll up, drill down and pan. We use the symbols defined in the Table 5.2 to explain how to generate the parameters according to the three data operations.

Concretely, drill down allows the user to navigate a more detailed view by either stepping down a hierarchical structure of a dimension or introducing additional dimensions. Algorithm 3 shows how to drill down the data resource based on the users’ input. Lines 2-12 check the dimension of the operation and assign the child of the current dimension to gain a more detailed view. If the dimension is changed, the dimensions used by the conditions also need to be changed correspondingly.

Roll up is the reverse operation of drill down - it performs the operation by climbing up a concept hierarchy for a dimension or by reducing the dimensions. Algorithm 4 outlines how to roll up the data resource based on the input from the user. Lines 2-5 perform an aggregation on the parent of the dimension. Similar as above, lines 7-11 indicate that the dimension used by the conditions also need to be changed to filter data at a higher level of granularity, for example, a filter by month or brand instead of a more fine-grained date and product with sales data.
Pan allows the user to change the data space he used to observe by changing the
dimensions of data or the operations used. If all the dimensions or operations are
traversed, the new condition which returns the remainder dataset will be presented.

Algorithm 5 illustrates how to pan the data resource given a service resource with
specified data operation and corresponding parameters. Assuming the chosen dimension and measures form a dataset, while the condition restricts the result to a
subset of the dataset. The output of a new condition will return another subset of
the dataset.

---

**Algorithm 3: Drill Down**

**Input:** \( R(O_p(D_o, M_t), [Cond(M_c : A, D_s : B), GroupBY(D_g)]) \)

**Output:** Updated \( R \) with new parameters

1. **Procedure** `DrillDown()`
2. for each \( d \) in \( D_o \)
3.     // check if the dimension is hierachical
4.     if `isHierarchical(D_o, d)` then
5.         \( D_o, d \leftarrow D_o, d, \text{child} \);
6.     end
7. end
8. for each \( d' \) in \( D \)
9.     if \( d' \) is not in \( D_o \) then
10.        \( D_o, \text{append}(d') \);
11. end
12. end
13. for each \( d^* \) in \( D_o \)
14.     if \( d^*, \text{changed}() \) and \( d^* \in D_s, d^* \in D_g \) then
15.         \( D_g, d^* \leftarrow D_o, d^* \);
16.         \( D_s, d^* \leftarrow D_o, d^* \);
17. end
18. end
19. return \( R \)
Algorithm 4: Roll Up

**Input:** $R(O_p(D_o,M_I), [Cond(M_c : A, D_s : B), GroupBY(D_0)])$

**Output:** Updated $R$ with new parameters

```plaintext
1 Procedure RollUp()
2     \textbf{for} each $d$ in $D_o$ \textbf{do}
3         // check if the dimension is hierachical
4         \textbf{if} isHerachy($D_o.d$) \textbf{then}
5             $D_o.d \leftarrow D_o.d.parent$
6         \textbf{end}
7     \textbf{end}
8     \textbf{for} each $d^*$ in $D_o$ \textbf{do}
9         \textbf{if} $d^*.changed()$ and $d^* \in D_s, d^* \in D_g$ \textbf{then}
10            $D_g.d^* \leftarrow D_o.d^*$; $D_s.d^* \leftarrow D_o.d^*$;
11        \textbf{end}
12    \textbf{end}
13 return $R$
```

Algorithm 5: Pan

**Input:** $R(O_p(D_o,M_I), [Cond(M_c : A, D_s : B), GroupBY(D_0)])$

**Output:** Updated $R$ with new Parameters

```plaintext
1 Procedure Pan()
2     let $\mu$ and $\mu'$ are all possible values for $M_c$ and $D_s$ respectively
3     \textbf{for} each $m'$ in $M_c$ \textbf{do}
4         \textbf{if} $m \notin M_I$ \textbf{then}
5             $M_I.m \leftarrow m', m \in M, m' \notin M_I$
6         \textbf{end}
7     \textbf{end}
8     $Cond(M_c : A, D_s : B) \leftarrow Cond(M_c : A', D_s : B'), A' = \mu - A, B' = \mu' - B$
9     return $R$
```

This resource navigation approach does not make any assumptions on how data should be explored. We merely provide recommendations based on historical inter-
actions and three data operation semantics. This satisfies the third requirement of process agnostic resource navigation in Section 1.4.

### 4.4 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 the scripts that can be applied to the data. For example, assuming 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. The definition of middle income might change over the years, thus, providing a script that implements the way of calculating middle income is more stable than providing a sub dataset.

#### 4.4.1 Data Package Structure

Fig. 4.7 gives an overview of the data format of the extended data package, which may contain:

- **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;

- **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;
• **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.

  – **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 4.3. The required properties are listed, others are omitted due to length limitation.

```json
{
    "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": "retrieve_good_employee_who_left",
        "path": "",
        "type": "python",
        "resources": ["good_employees_left", "."]
    }],
    "provenanceLogs": [{
        "lastPackage": "",
        "created_in": "06/12/2017"
    }]
}
```
4.4.2 Data Package as a Resource

As a data analyst makes her data exploration by issue and refine queries on a sequence resources, until find the satisfied data. Data packager can be used to supply her with a function to record her query session and formulate it with data result to a data package. As introduced in the batch conversation model and Section 4.2.1, 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). The data package service serves data package representation based on its semantic protocol as Fig. 4.8

4.4.3 Data Package as a Context Service

Data package also plays a role of providing the data context information that informs the historical usage of the data, the upstream lineage, the data constraints like privacy compliance policy, etc. 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 and decide if it can be chosen to do her analysis later. Another example of data context information is that provenance property recording the package chain linked by the hypermedia to trace the track of data usage in the data pipeline.

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. This satisfies the forth requirement of sharing data processing and context information in Section 1.4.
4.5 Conclusion

Existing data services provide data retrieval interfaces for data analytics in many practices. However, they have limitations in supporting data exploration and data retrieval due to their one-off interaction and lack of guidance in data exploration. Moreover, due to their one-size-fits-all characteristic, the data-centric web service cannot meet the requirement of data analytics using interactive, stream and batch data retrieval patterns.

In order to fill the gap, we proposes a REST-based data service API design for data exploration as a service (DEaaS), which specifies data retrieval interface targeting data analytics. DEaaS creates and normalizes a set of data and service resources for data exploration. Following the HATEOAS constraints of REST, a service navigation approach is proposed, we leverage a discovery tree which is constructed from the history queries to match the user trace and discover the appropriate resource API. The parameters of API are recommended according to the different semantics of data operation. 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 resource navigation consists of three main components, where the metadata service and discovery tree are not restricted by the type of underlying data sources. The parameter recommendation is mainly based on the data analytics semantics which is tightly bound to the underlying multidimensional data model. However, this solution can also be applied to other data types and models, like semi-structured data or graph data model, as long as the semantics are analyzed and defined in the protocol.

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.

There are three main benefits of using our data service to realize data exploration. First, pushing computation to the data side is more efficient than pulling the whole data out for computation. Second, from the data publishers’ perspective, they can expose subset of 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.

DEaaS facilitates human-in-the-loop processing needed for interactive data exploration and brings domain knowledge into data service design. By automatically recommending data query and streamline the data exploration, performance is improved in an exploratory analysis.
Chapter 5

Evaluation

In this chapter, two case studies and one experiment were conducted to do a qualitatively and quantitatively comparative evaluation of the proposed data service design against the OData REST design, which is an OASIS standard protocol that defines how to build the RESTful APIs for open data. This chapter is organized as follows. Firstly, a API framework prototype is implemented and described in Section 5.1; Then, a case study in human resource scenario is conducted on an open simulated dataset to evaluate our design on four metrics including REST maturity, interoperability, discoverability, and reusability in Section 5.2; To avoid the threaten to external validate of the approach, another case study of household travel scenario is conducted on a real-life dataset to evaluate usability and adaptability in Section 5.2.4. After that, an experiment is conducted to evaluate the performance of our approach in Section 5.3; Finally, Section 5.3.4 discusses the validity of the case studies and conclude this chapter.
5.1 Prototype Implementation

The RESTful web services were developed using a Java framework called Jersy\(^1\). Java was selected due to its popularity in building web services, platform independence, ease of use, and reliability. JAX-RS\(^2\) is a Java API library for creating RESTful web services, which uses Java annotations to map an incoming HTTP request to a Java method. Jersy is the JAX-RS Reference Implementation from Sun to simpler development of RESTful Web services for developers. All data returned from the data services are in JSON which has become de-facto message format for data interchange on the web due to its standardization and simplicity.

The resource navigation is implemented in the backend system by enriching Linked Description Objects (LDOs) in JSON Hyper-Schema\(^3\). PostgreSQL\(^4\) was chosen as data storage layer, other databases like SQLite\(^5\) are also supported as long as they are equipped with JDBC driver.

DEaaS was built with Spring Framework. Specifically, referring to the DEaaS architecture in Fig. 4.4, the main component was implemented in the spring Framework as shown in the Fig. 5.1: the resources were implemented in the service layer as \textit{endpoints} and resource navigation was implemented as \textit{APISuggestionServices}.

The repository layer is responsible for communicating with the data storages. The domain model incorporates both data entities and related behaviors on the data. Data Transfer Object (DTO) is used to batch up the parameters of multiple call into a single object for transferring data easily. Several DTOs were defined for the response message to contain the resource navigation information.

\(^1\)\url{https://jersey.github.io/}  
\(^2\)\url{http://cxf.apache.org/docs/jax-rs.html}  
\(^3\)\url{http://json-schema.org/latest/json-schema-hypermedia.html}  
\(^4\)\url{https://www.postgresql.org/}  
\(^5\)\url{https://www.sqlite.org/index.html}
5.1.1 Repository Layer

The implementation of repository layer mainly contains a couple of Java classes and interfaces for data access model which is a connector to the database. As shown in Fig. 5.2, the dataRepository interface has all methods to access the underlying data source and pull the required information, while JdbcDataRepositoryImpl is the class implementing dataRepository, which executes SQL script to fetch the results. Referring to the data package semantic in the Fig. 4.8, the DataPackageRepository interface and PackageScriptRepository interface define the methods to create and get the package and scripts respectively. The MemoryPackageScriptRepositoryImpl class implements the DataPackageRepository to return the created package ID and content of the package. The MemoryPackageScriptRepositoryImpl implements the PackageScriptRepository to create the script ID and acquire the script content.
by the script ID. In this implementation, the package and scripts are created in the memory temporarily, alternatively, the packages and scripts can be stored on the remote storage or cloud.

5.1.2 Service Layer

In the service layer, as shown in Fig. 5.3, the DEaaS resources including gateway, sampler, statisticFunction, aggregator and packager, were all implemented in the DataExplorationEndpoint class. In the implementation, the HTTP methods like GET, POST and OPTIONS are used to map the defined operations on each resource. For example, as shown in Listing 5.1, @Path is used to indicate the different resource on the same dataset, which is regarded as parameter for each resource. @GET is used to retrieve the requested data. @OPTIONS is used to describe the resource and specify how to request the resource. @POST on the packager generate new package where the exploration history on the dataset is generated as scripts by PackageScriptGenerationService for each user session.

The ApiSuggestionServiceImpl implements ApiSuggestionService to recommend the resource and parameters for the next call of each API. Concretely,
the discovery tree algorithms in Section 4.3.2 are implemented and encapsulated as a tree web service. In the `ApiSuggestionServiceImpl` class, the discovery tree is built and updated by PUT operation on the service `http://0.0.0.0:3000/update_tree/` using API call history, and the recommendation is returned from the service `http://0.0.0.0:3000/next_task/` by POST operation.

@GET
@Path("{dataset}/filter")
@Produces(MediaType.APPLICATION_JSON)
public Response filter(@Context UriInfo uriInfo,
@HeaderParam("user-session-id") String userSessionId,
@PathParam("dataset") String dataset,
@QueryParam("dimensions") List<String> dimensions,
Apart from the endpoint for the main resources, the two endpoints for package and scripts were also defined as independent resources from the specific dataset. Through these two endpoints, the package and scripts can be arbitrarily customized with any inline or online datasets by the user.

5.1.3 Domain Model and DTOs

Domain model specifies all the data objects needed in the application. This application involves resource objects and data package object. In this implementation, as shown in Fig. 5.4, in order to make the service layer agnostic to the underlying dataset. The detailed attributes of dataset are not specified in the DataResource, instead, enum type was used to list the predefined service resources to expose the features of the data resources for analytics purpose. Corresponding to the data package structure in Fig. 4.7, the DataPackage class is associated with the classes of
Figure 5.4: Domain Model Classes

PackageDataSource, PackageScript, packagePrivacy, and packageProvenance.

The related package information are defined in the corresponding classes.

In Spring Framework, DTO is an object that carries data between processes.
Instead of performing many remote calls for application, an assembler of the parameters is used on the server side to transfer data between the layers of API framework. A set of DTOs were defined for the API call response based on the design requirement as shown in Fig. 5.5.
5.2 Case Study

According to the case study methodology discussed in Section 3.3.1. The case study was conducted following several steps, including:

- **Step 1** — Designing case study objectives and specify the cases to be studied;
- **Step 2** — Preparing for the protocols and procedures for data and evidence collection;
- **Step 3** — Collecting the case-specific data and evidence;
- **Step 4** — Analysing the collected data and evidence.

The remainder of this section discusses above steps in detail: Section 5.2.1 discusses the research objective, research questions and the rationale for the selection
of the cases. Section 5.2.2 explores the metrics used in the case studies and human subjects protection. The evidence collection and analysis of the two case studies are discussed separately in Section 5.2.3 and Section 5.2.4.

5.2.1 Case Study Plan

To exercise this solution design and validate its feasibility, case studies were conducted to do a comparative evaluation of the proposed data service design against the OData REST design with respect to a list of metrics, including REST maturity, interoperability, discoverability, reusability, usability and adaptability. More details of metrics are discussed in Section 5.2.2. OData is a REST-based OASIS protocol for querying and updating data, which is built on technologies like HTTP, ATOM/XML and JSON. It provides a uniform and standardized way to describe the data and the data model for easier interoperation between data sources, applications, services and clients. Thus, this case study uses OData as the comparison subject.

In case study, the case and the corresponding units of analysis should be selected intentionally [61]. The purpose of selection is to study a case that is expected to be typical and representative. The HR analytics scenario discussed in Section 1.3 is from Kaggle, which is an open data platform for data analytics community to explore and produce predictive models on the open datasets. One open dataset and related kernels (data processing scripts) are selected for the HR case study. The kernels all have the same purpose to find why some of most experienced employee are leaving prematurely. The key difference between the analytics processes is the sequence or selection of data operations.

Another case is chosen from Australia transport census scenario. Household
Travel Survey Dataset (HTS) [5] is a data layer of Australia transport census integrated from multiple data sources with different formats, hosted by the Australian government. The HTS project aims to create a common data layer of Australia transport census, where users can obtain insights into mobility patterns and utilization of public and private transport. The purpose of the data exploration processes on this data layer includes discovering relationships between travel choices, exploring how transport infrastructure could be improved, understanding how travel choices are influenced, and improving travel outcomes.

5.2.2 Preparation for Case Study

A case study protocol should be developed and refined in the preparation stage, which specifies the prior techniques and procedures for collecting the case study evidence in the following steps.

Collection Techniques

Through the case studies, data were collected from the available work artifacts during the design and implementation, including the system design, process specification, and the code implemented. In terms of comparison with OData design, XOData⁷ was selected as a tool for rapid prototyping the OData design in the selected case scenario. XOData is a generic OData API/Service visualizer and explorer, which assists in rapid prototype, verification, testing, and documentation of OData APIs. XOData provides interactive model-diagram of datasets for API/data-model awareness. OData APIs can be quickly and easily built using XOData Chrome App. All these features assist the case study preparation for comparative evaluation.

Exploring Metrics

The definition of what data should be collected is based on Goal/Question/Metric (GQM) method [16] [70]. In GQM, the measurement should be defined in a top-down fashion, starting from specifying goals, then tracing the goals to all life-cycle products, processes and resources, and finally interpreting the collected data with respect to the goals, context characterization and environment. This means the metrics are derived based on the goals that are formulated for a specific measurement activity. It also implies that the researchers can control the quality of the collected data so that no unnecessary data is collected [61]. In this subsection, following the GQM model, the goal is established first, then the questions are derived from the goal. Lastly, the metrics used in the comparative analysis of the two case studies are explored based on the questions.

Functionality and quality attributes are two orthogonal perspectives of a software requirement [55]. The functionality describes the software ability to do the work that is intended to be done. Software quality attributes, on the other hand, provide benchmarks to measure the suitability of the intended behavior of the system within the environment where it is built. Achieving quality attributes are considered throughout the whole life cycle of software development, including design, implementation, deployment and runtime. In this study, referring to the research question to design RESTful data services to make the data retrieval efficient and effective for data analytics, the purpose of measurement is to compare the usability of the system implemented by proposed API design against the OData design. Usability concerns how easy it is for the user to accomplish a desired task and the kind of user support the system provides. This research mainly focuses on the area of adapting the system to user needs. Specifically, the system may automatically fill in URLs based on a user’s past entries.
Based on the goal specified above, several questions were derived to characterize the goal into metrics for the case studies.

Question 1. Does the system align with the REST principle?

Question 2. Does the system provide meaningful information during each interaction?

Question 3. Does the system guide user to discover the resource needed?

Question 4. Does the system support multiple scenarios or datasets?

The metrics were proposed to answer the above questions. In terms of the first question, the Richardson Maturity Model\textsuperscript{8} is a measurement to evaluate how well the web services adhere to REST principles. Corresponding to the Second question, interoperability describes the ability to which two or more systems can exchange useful and meaningful information via interface in a particular context. The definition does not only refer to the ability to exchange data syntactically but also correctly interpret the data being exchanged. Achieving interoperability involves the relevant systems locating each other and managing the interface to exchange information \cite{55}. Referring to the third question, discoverability can be a metric to measure if the system provide meta-information recorded against each available service that needs to be consistent and meaningful for user to discover. Corresponding to the last question, adaptability refers to the ability of a system to cater for different computing environment dynamically. More details about metrics are discussed in Section 5.2.3 and Section 5.2.4 separately.

Data Collection Procedures

Table 5.1 summarizes the data collection procedure of the two case studies, which provides a guidance for the following data collection and analysis. Both case stud-
Table 5.1: Data Collection Procedure

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Case Study One</th>
<th>Case Study Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Maturity; Interoperability;</td>
<td>Usability; Adaptability</td>
</tr>
<tr>
<td></td>
<td>Discoverability; Reusability</td>
<td></td>
</tr>
<tr>
<td>Collection Technique</td>
<td>Observation; XOData</td>
<td>Observation; XOData</td>
</tr>
<tr>
<td>Data Collected</td>
<td>Resource representation;</td>
<td>Proof-of-concept prototype;</td>
</tr>
<tr>
<td></td>
<td>Response message;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>data package;</td>
<td>Development procedure</td>
</tr>
</tbody>
</table>

ies followed the comparative methodology, which was used to compare OData and this design under the same scenario. The evaluation of this research extended the comparative methodology with more measurable metrics to compare the alternative approaches in perspective of the API representation design as well as the implementation of the produced systems.

In the first case study, the explorative analytics processes in the HR scenario were collected from Kaggle\(^9\) and then simulated using the developed data service APIs in Section 5.1. During this process, the resource representations were checked if the provided service resources could satisfy all the analytics operations in the data exploration processes. In addition, to validate the interoperability and discoverability, the recommended resources in the response messages were collected to matches the operation that the analyst actually executed in each kernel. In order to evaluate the reusability, the returned data package is also collected to execute to recur the exploration process and check if the same result is returned. To compare with

---

\(^9\)https://www.kaggle.com/
Chapter 5

OData design, the explorative processes in this scenario is also implemented using XOData. The related resource representation and messages are collected, analyzed and compared based on the selected quality attributes.

The second case study was conducted on the HTS project sponsored by Data61, CSIRO. All the required analytics operations are derived from the Australian Urban Research Infrastructure Network (AURIN)\(^\text{10}\) which provides a framework for researchers to access, investigate and use a wide range of data from across Australia [66]. The usability was validated through the application of the prototype to this real-life scenario. Similarly, the same operations are also implemented by OData design. The development procedure, which were implicated by the implementation code and related configure settings, were collected from the produced system separately. The adaptability was validated to compare the development procedure against the OData implementation in the same scenario.

### 5.2.3 Case One: HR Analytics

#### Objective

The objective of the first case study involves four aspects. The first is to measure the REST maturity of the DEaaS APIs. The second is to evaluate the interoperability of the system to support the client-server interactive data exploration. Then, the discoverability is validated for service discovery. Lastly, reusability is also considered to support reuse of the data exploration process.

#### Case Description

Fig. 5.6 shows the roadmap of a typical data exploration process in HR scenario with the alternative relations and the resources. The purpose of this data explora-

\(^{10}\)https://aurin.org.au/
ration process is to understand why good employees left a company. Based on the data exploration process, data analysts start 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 percent-
age 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.

Case Implementation

GET /HRAnalytics/gateway

URIs: /HRAnalytics/(resources)

OPTIONS: /HRAnalytics/functionSupplier

URIs: /HRAnalytics/functionSupplier/(name)

GET: /HRAnalytics/functionSupplier/isnull/all

200 URI: /HRAnalytics/sampler

GET: /HRAnalytics/sampler?size=100

200 URI: /HRAnalytics/aggregator

GET: /HRAnalytics/aggregator?attributes=job,left&fields=left&operation=ratio&groupby=job

200 URI: /HRAnalytics/filter

GET: /HRAnalytics/filter?left==1,satisfaction_level>0.7,time_spend_company>=4,time_spend_company<=6

200 URI: /HRAnalytics/packager

POST: /HRAnalytics/packager/(sessionID)

201 created /HRAnalytics/packager/{packageID}

Figure 5.7: An data exploration process with DEaaS interactions on HR dataset
Corresponding to the data exploration process described in the previous section, the DEaaS data services interactions is shown in Fig. 5.7. Specifically, for the sake of easy understanding, a snippet of the code represents the interactions during the exploration as shown in Listing 5.2. Apart from the status code and data results, the recommended resources and parameters are returned in each step.

GET http://data_service.com/HRAnalytics/gateway
Response:

```
{
    "DataStatus": "static",
    "DataSize": 14999,
    "Attribute": {
        "name": "string",
        "age": "int",
        "salary": "int",....
    },
    "Description": "Employees salary of Australia",
    "Resources": {
        "Aggregator": "http://data_service.com/HRAnalytics/aggregator",
        "Filter": "http://data_service.com/HRAnalytics/filter",
        "Sampler": "http://data_service.com/HRAnalytics/sampler",
        "Functions": "http://data_service.com/HRAnalytics/functionSupplier",
        "Packager": "http://data_service.com/HRAnalytics/packager"
    }
}
```

Request:
GET http://data_service.com/HRAnalytics/aggregator?Attributes=left&Fields=left&Operations=ratio
Response: 200 OK

```
{  
    "value": {...},  
    "_links": [{  
        "rel": "drill down",  
        "href": "http://data_service.com/HRAnalytics/aggregator?Attribute=job, left&Fields=left& Operation=rtio&Groupby=job"
    },{  
        "rel": "roll up",  
        "href": "http://data_service.com/HRAnalytics/aggregator"
    }
```
Listing 5.2: A snippet code of data exploration for HR Analytics

Maturity

The Richardson Maturity Model\textsuperscript{11} was applied to evaluate how well DEaaS 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. DEaaS 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, DEaaS data service APIs achieve the highest level of maturity of REST.

\textsuperscript{11}\url{https://martinfowler.com/articles/richardsonMaturityModel.html}
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 discovering services and handling of response from service requestor [22]. 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 [67].

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 5.2 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.

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

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 Section 5.2.3, an aggregator API to group the percentage of left employee by job can be recommended for client who requested the percentage of left employee.
Table 5.2: Comparison of information sharing at Semantic level

<table>
<thead>
<tr>
<th></th>
<th>OData Service</th>
<th>DEaaS Design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analytics</strong></td>
<td>• Mapping to the underlying data schema</td>
<td>• Conforming to data analytic operations</td>
</tr>
<tr>
<td><strong>operation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Analytics</strong></td>
<td>• User driven</td>
<td>• Definition of analytical relations in HATEOAS</td>
</tr>
<tr>
<td><strong>process</strong></td>
<td>• Manually constructing</td>
<td>• Intelligent recommendation</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>• Manually collected by data users</td>
<td>• Data package chain</td>
</tr>
<tr>
<td><strong>information</strong></td>
<td></td>
<td>• Record data exploration process</td>
</tr>
</tbody>
</table>

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.
DEaaS data services propose a roadmap of data exploration that defines different relations in links property so that a service can be reached from different resources. In addition, HTTP OPTIONS was used 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. The HATEOAS with semantics enables auto-discovery for applications and presents the services as a graph illustrated in Fig. 4.5.

The Discoverability makes it possible to automate service interactions. Compared with the general-purpose search API like Twitter REST API, DEaaS 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. In the first case study, when the data analyst send a GET request and run the scripts that the data package created from HR data exploration process, the previous data operations can be reproduced. 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.
5.2.4 Case Two: Household Transport Survey

Objective

The objective of the second case study is to evaluate the usability and adaptability of the system by applying for the real use case and catering for the different data environments respectively.

Background

The Australian Urban Research Infrastructure Network (AURIN)\textsuperscript{12} is a national project which provides researchers, government and industry across Australia with a secure, nationally accessible Internet portal to access datasets from participating Australian states and information services. The datasets include census results (e.g. demographic and socio-economic profiles), geographic data (e.g. location of roads, rail, and other infrastructure), and organizational data (e.g. Commonwealth, State, Local organizational structures, businesses, and hospitals) among others. Furthermore, AURIN is also a platform which provides a set of visualization and analysis tools for researchers to access, explore and analyze the provided data [66].

The targeted dataset for the second case study is a household travel survey dataset (HTS) which is collected by local government associations with different schema and harmonized by Australia’s Information and Communications Technology (ICT) Research Centre (NICTA) on 2013. The survey provides data on the numbers, lengths and purposes of trips made by different means of transport (walk, cycle, bus, car, train). The HTS dataset does not only contain current information about the demographic, socioeconomic, and trip-making characteristics of individuals and households, but also enable researchers to explore and discover new knowledge around Australians mobility patterns like choice, location, and scheduling of

\textsuperscript{12}https://aurin.org.au/
daily travel. It would allow planners to investigate for example, how transport infrastructure could be improved, discover relationships between travel choices, determine how travel choices are influenced, and might even allow for improvement of travel outcomes.

**Case Description**

The analysis use case is to develop the travel forecasting model and to predict changes in daily travel patterns in transportation systems and services. HTS plays a role in evaluating changes in transportation supply and regulation. Specifically, some key analysis of the data include describing patterns, for example how different groups of people travel, monitoring trends in travel, and assessing the potential equality impacts of transport policies on different groups.

AURIN provides a set of data visualisation and analysis tools. Data analysts can explore the dataset and do the analysis modeling on this platform. To conduct the case study, four data scientists were invited to conduct explorative analysis on the HTS dataset on the AURIN. To give an example of operation, a snapshot of AURIN output is presented in Fig. 5.8, which shows the population on 2011 by local government association (LGA).

Similar to the case study one, seven processes were simulated using DEaas APIs and OData APIs respectively. A representative example of data exploration process for this analysis use case on HTS is as shown in the Fig. 5.9. The data scientist first need to have an overview of the dataset. This requirement can be satisfied by calling the gateway which returns the metadata and the provided service resources. The response is shown in Listing 5.3. From the returned results, the data scientist acquired the categorical and numerical features. Next, the data scientist started to aggregate the numeric value by the categorical feature to further understand the
distribution of each feature using aggregator. Listing 5.4 indicates how to call the aggregator APIs. After this, the data scientist sends a GET request to function supplier to check the missing value of each attributes. Then he normalised the data to remove the outliers, for example, the “-1.000” in this dataset. After the data normalisation, data were scale in one level, then the data scientist continued to call function supplier to acquire the correlation matrix to observe the relationship between features. Lastly, a POST request was sent to packager to create this whole data exploration process.

```json
{
  "description": "hts_data",
  "dataStatus": "static",
  "dataSize": 214,
  "attributes": {
    "area": "varchar",
    "total_households": "int4",
    "total_vehicles": "int4",
    "areacode": "varchar",
    "av_people_per_household": "float8",
    ...
  }
}
```
1. Gateway
Acquire the metadata of the dataset

2. Aggregator
A summary of the numeric data on the categorical value

3. Function Supplier
Check the number of the missing value and replace them

4. Function Supplier
Data normalisation to remove the outlier

5. Function Supplier
Make a correlation matrix to observe the attributes relationships

6. Packager
Record the data exploration process and return the data package

Figure 5.9: An HTS Analytics Data Exploration Process

```
"total_bicycles": "int4",
"md_people_per_household": "float8",
"md_vehicles_per_household": "float8",
"survey_year": "varchar",
"population_persons": "int4",
"av_bicycles_per_household": "float8",
"md_bicycles_per_household": "float8",
"av_vehicles_per_household": "float8"
},
"resources": {
  "filter": "http://localhost:8080/hts_data/filter",
  "aggregator": "http://localhost:8080/hts_data/aggregator",
  "function": "http://localhost:8080/hts_data/function",
  "packager": "http://localhost:8080/hts_data/packager",
  "sampler": "http://localhost:8080/hts_data/sampler"
},
"userSessionId": "2437d4752-7979-6c7a-1b46-
cac2bd4ef93478cc75e175b6b99d0cbce501e23758"
}
```

Listing 5.3: A snippet code of data exploration for HTS Analytics
"path": "http://localhost:8080/hts_data/aggregator",
"method": "GET",
"parameters": [
{
"name": "operation",
"description": "The aggregation operation to apply (e.g. avg, sum)",
"required": true,
"type": "string",
"paramType": "query"
},
{
"name": "attributes",
"description": "The data attributes to apply the aggregation to",
"required": true,
"type": "string[]",
"paramType": "query"
},
{
"name": "groupBys",
"description": "The data attributes to group the aggregation by",
"required": false,
"type": "string[]",
"paramType": "query"
},
{
"name": "filterCond",
"description": "If specified, filter the records by this condition before aggregation",
"required": false,
"type": "string",
"paramType": "query"
}
],
"responseMessages": [
{
"code": 401,
"message": "Missing or invalid user session ID specified."
}
]
Usability

Usability refers to the degree how easy it is for the user to accomplish a desired task and the kind of user support the system provides [22]. The second case study demonstrates the usability of DEaaS for the design and implementation of the data exploration process in the real-life data environment.

The runtime tactics for supporting usability involves in supporting user initiatives and system initiative [10]. According to each response during the interaction between the user and DEaaS, the system can support the user’s requirements efficiently by providing the guidance for the user. In contrast, the OData service resources are isolated. In order to find the suitable resource for each step of data exploration and preprocessing, the user had to traverse all the resources or resort to tutorial or documentation. In terms of system initiative, the system needs to contain the information used to predict the system’s behavior or user’s intention. Tailoring and modification can be either dynamically based on past user behavior of offline during development [22]. The navigational model of DEaaS records the API call history based on each user session, and ingests them as input for the resource recommendation. The resource navigation model plays the role of task model to determine the context so that the system have the idea of what the user is attempting and provide assistance. In addition, the user model is also maintained in DEaaS by analyzing the data operation patterns and presenting the semantics between operations for users. In contrast, OData does not provide any task model
Adaptability

Adaptability refers to the ability that a system may be changed to fit new requirements \cite{55}. Services are being developed must dynamically adapt to different computing environments, which means the interactions between the services and the underlying infrastructure must be managed properly.

In our case study, the OData services need to be re-developed for HTS datasets, because their query parameters in the API representation are tightly bound to the table schemas of the underlying data resources as indicated in Fig. 5.10. When the table schema changes, the implementation of the query to the data repository have to be changed correspondingly.

On the other hand, DEaaS adopts a multidimensional data model to unify and normalize various data formats. The DEaaS API representations map to this unified data model. As indicated in Fig. 5.10, the query parameters of API in DEaaS are consistently bound to the normalized representation of the data model (e.g. dimension and measure) rather than specific tables or attributes, which are different from OData representation. In practice, DEaaS service resources adapt to both HR and HTS datasets seamlessly without second development thanks to the same data model representation. Furthermore, DEaaS services are applicable to any datasets, given that the data are normalized to the multidimensional data model.

Limitation

Compared to exploring dataset on AURIN platform, the DEaaS APIs cannot achieve the same level of flexibility and data visualizationm. The exploration and
preprocessing operations provided by function supplier in DEaaS are not as abundant as by AURIN. In addition, DEaaS APIs cannot support data visualization independently. In some cases, the charts like histogram and box plot are very necessary to data exploration. DEaaS need to integrate the web client like D3\footnote{https://github.com/d3/d3} to implement the function of data visualization.

The navigational model is restricted by the experience and knowledge of the data scientists. Data scientists tend to apply different operations to explore and preprocess the same dataset using data service APIs, as a result, the recommended resources by the discovery tree may not be the optimal choice for the data scientist.

## 5.3 Experiment

### 5.3.1 Objective

Corresponding to the research aim to design data services for efficient and effective data retrieval, the objective of the experiments is to measure the performance of the resource navigation approach of the DEaaS. The performance of the system can be measured by two factors, the efficiency and the effectiveness. The efficiency

```plaintext
/*HR dataset*/
DEaaS: /hrAnalytics/filter?dimension=id,name&measure=salary
OData: /hrAnalytics/Persons/?$select=id,name,salary

/*HTS dataset*/
DEaaS: /hts_data/filter?dimension=area_code,population_persons
OData: /hts_data/population?$select=area_code,population_persons
```

Figure 5.10: An example of API representation comparison
indicates the ratio between the inputs and outputs which are applied in the system. Being efficient means the system uses inputs in a ‘right’ way. The effectiveness is the measure for deciding whether the system provides the desired output or not. Being effective means producing the right output in terms of quantity and quality. When the system is ineffective, the system is out of control and it needs a major correction. A system has to be effective and efficient for the highest utility to the user of the system.

5.3.2 Experiment Design

The experiments were conducted on the prototype of API framework Section 5.1 on an Intel Core i5 with 8 GB RAM, running OS X 10.10.5. The OData services are implemented with the same configuration but without navigation mechanism following the specification of OData 4.0.

Throughout the experiments, the associated analytics processes on below two datasets are simulated using the build data services.

- **Human Resources Dataset (HR)** is a synthetic dataset of employee information which is a CSV file with 21 attributes and 15,000 rows. In Kaggle, data scientists compete to produce the best models for exploring public datasets and predicting the results. 8 explorative analytics processes were chosen on this dataset with the same purpose to understand why good employees left a company. Apart from the status code and data results, the recommended resources and parameters are returned in each step. DEaaS and OData executed the processes as 8 query sessions with 217 queries and 404 queries respectively.

- **Household Travel Survey Dataset (HTS)** is a dataset with 70 attributes scattered in five tables with a maximum of 20,000 rows. 7 data exploration logs derived from the HTS project sponsored by Data61 Australia were chosen, then
were used to reenact the processes by DEaaS and OData respectively. DEaaS and OData executed the processes as seven query sessions with 252 queries and 397 queries respectively.

In order to evaluate whether the resource navigation can find the targeted attributes and data segments, HR-1 and HTS-1 were used as the original datasets and half feature size of both datasets were reduced to form HR-2 and HTS-2 respectively. The same data exploration processes were simulated on the reduced datasets and investigated how the performance change with the different feature size. As a result, DEaaS and OData contain 119 queries and 308 queries on HR-2 respectively, whereas 112 queries and 259 queries on HTS-2 respectively.

5.3.3 Performance Metrics

In information retrieval, a set of commonly used performance metrics are always adopted to measure the efficiency and effectiveness of the approach, including Precision, Recall, and F-score.

To formally define the metrics in our experiment, $R_r$ is assumed as the number of the relevant resource APIs which compose each data exploration process, and the $R_t$ as number of resource APIs retrieved by Service Navigation. $R_p = R_r \cap R_t$ represent the number of subset of relevant service in the retrieved set.

$$Precision = \frac{R_p}{R_r} \quad (5.1)$$

measures the ratio of number of recommended relevant resources to the relevant ones.

$$Recall = \frac{R_p}{R_t} \quad (5.2)$$

measures how many positive true values in the returned resources.

$$F\text{-}score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \quad (5.3)$$
represents a harmonic mean of precision and recall, where F-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

**Results and Analysis**

The F-scores of DEaaS APIs and OData APIs on each data exploration process of HR dataset are shown in Fig. 5.11(a). The trendlines of F-score on both API sets that target seven explorative data analysis on HTS datasets are shown in Fig. 5.11(b). Both figures show that F-Score of DEaaS API increases as the historical processes increase, while F-Score of OData API keeps in a range of relatively low value. This indicates that the discovery tree could improve the accuracy of service recommendation with an increasing number of users’ request.

In some special cases where analysts explore in an abnormal way, the F-score can drop (see the third value of the red line in Fig. 5.11(a)). Since DEaaS resource recommendations rely heavily on historical queries, this drop is to be expected. If other analysts follow suit and this abnormal way becomes the norm, the F-Score will increase.
Through the comparison of the average numbers of queries using two approaches, we can observe that DEaaS approach outperforms DaaS in exploring both the HR and HTS datasets. From HR-1 and HTS-1 of Fig. 5.12, we can see that the average query number in DEaaS is much less than that in DaaS. Furthermore, when we adjust the size of variables of both datasets, as shown in HR-2 and HTS-2 of Fig. 5.12, the gap becomes even greater. This demonstrates that users can figure out the targeted variables more quickly when being guided by the parameter recommendation.

5.3.4 Validity

A possible threat to external validity of the experiment approach is to use small datasets [57]. The performance of query execution like response time or latency
may be affected if we scale up datasets. However, this performance is also related to the system implementation and other factors like infrastructure and network, which is out of the scope of this thesis. This thesis mainly focus on service design and recommendations. The performance of service recommendation depends on the feature size of datasets rather than the massive number of instances. We made an effort to find the real-life dataset that contained sufficient features and evaluated our approach in terms of different feature size in Section 5.3.2.

5.4 Conclusion

The proposed solutions have been evaluated through case studies and experiments. The case studies were conducted by following a systematic approach. The human resource scenario used was selected from Kaggle and the household travel scenario from AURIN. The case studies show that 1) DEaaS data service APIs adhere to REST principles, reaching at the highest level of REST maturity model; 2) DEaaS data services have advantages in interoperability and discoverability by interpreting of analytics domain operation and sharing of data context information; 3) Data package as a media type of data service is an effective method for reusing and sharing the derived data and processing scripts among analysts; 4) DEaaS data services are adaptable to different data environments dynamically without second development. The experiment was conducted in comparison with OData design. The experimental results demonstrates that the proposed resource navigation approach can make DEaaS outperform existing data services in data exploration.

During the case studies and experiment, several limitations of this work were reflected, which will be improved in the future work. Firstly, in this prototype, the resources provided by DEaaS are not comprehensive enough and need comple-
mentary tools to achieve data visualization. However, the DEaaS architecture is scalable and more resources can be developed on demand in the future; Secondly, the discovery tree is subject to the experience and knowledge of the data scientist, because it only recommend the resource following the most popular exploration process arbitrarily, which may not satisfy some special users who explore data in an abnormal way; Thirdly, the parameter recommendation mainly depends on the data analytics semantics which is derived from the underlying multidimensional data model, while other data types and models, like semi-structured data or graph data model need to be considered as well. We will analyze more data models and define their analytics semantics for the protocol in the future work.
Chapter 6

Conclusion and Future Work

As the era of big data is soon to arrive, decision-makers from a range of different domains would like to make decisions and take actions based on big data. Accordingly, data analytics has become a hot discipline and has drawn attention from both industry and academia. Data analytics is the process of acquiring, organizing, and analyzing large sets of data in order to discover and visualize patterns and other useful information[69]. As data on the web become increasingly larger and more complex, the questions of how to efficiently extract these data for analytics is becoming a pressing issue [44],[25].

However, at present, data is still locked behind various applications and stored in multiple systems, especially in the government environment, where some valuable data exist in all kinds of digital forms, hides in highly secured databases, and are locked behind firewalls. In the interests of convenience, many agencies have resorted to simply publishing this data on web pages, or even to some data sharing platforms in the form of PDF files and CSV files. In order to acquire and interact with these data, any potential data consumer is required to handle multiple interfaces or mechanisms[14]. Web crawling and manually bulk-downloading from ad-hoc
websites are two mainstream methods used in this environment.

The emergence of REST and the wide-spread usage of API has enabled business to process and access data regardless of wherever it resides. Data services, which are also adopted by some enterprises as a better strategy, focus on providing an uniform access to data for their clients by exposing data so that it is can be easily accessed over simple access interfaces, bypassing the application logic layer. Furthermore, data services support application development to integrate data services into their service-oriented applications. In the data analytics area, data services provide uniform, scalable, and filtered interfaces for data analysts to retrieve data [14]. Many websites and platforms, such as Twitter, Google and CKan\(^3\), offer data service APIs that provide simple, easy-to-use access to some of their resources, thus enabling third parties to integrate the data service into their own service-oriented applications.

However, the existing “one-size-fits-all” data service API design is inefficient and ineffective in terms of supporting different data retrieval patterns such as interactive, batch, and stream query. The one-off API query representation focuses on underlying data sources rather than data analytics needs, without leveraging hypermedia and data packaging techniques to help users navigate data exploration. Moreover, some organizations view their data as an asset and are unwilling to open up the entirety of the raw data due to issues relating to privacy and commercial competition. On the other hand, data analysts desire to have access to the original raw data, along with the most features possible, to ensure prediction accuracy. Accordingly, the trade-offs between data accessibility and data constraints and between the needs of the data provider and data consumer are important issues in data service design.

\(^3\)https://ckan.org/
As data exploration is a labor-intensive and sometimes repetitive task for analysts, it could be beneficial to circulate this value-added data among the analytics community. To this end, data consumers can reproduce and share results from earlier exploration operations in order to streamline the data analytics pipeline. Thus, the issue of how to provide provenance detailing what earlier manipulations have been performed on the data, so that data consumers are more informed on the assumptions contained in the data and can conduct the appropriate analysis later, should be considered in data service design.

6.1 Summary of the Work

The present research follows the empirical research methodology based on the guidelines provided in [50] (with some customization), which combines rigorous research with real-world cases so that the relevance of the theoretical contribution can be examined using empirical data. Specifically, we began by analyzing elements of data analytics APIs, including data format, method structure, data model, data processing operations, and usage policies, then conducted a thorough review of REST, HTTP, and data packages to match the REST constraints on data service APIs for data analytics. In the following step, a RESTful API design for data analytics was proposed to address the research questions in Section 3.1, which focus on a data retrieval interface targeting data analytics. The main contributions of this work are as follows:

1. Based on the analysis of data retrieval patterns (i.e. batch, interactive, and stream) in data analytics, a set of RESTful conversation models are proposed to depict the interaction between resources for different data retrieval patterns, incorporating a negotiation protocol to reconcile the data analysts
requirements with data providers privacy policies.

2. Motivated by the requirements analyzed in Section 1.4, a three-tier RESTful service architecture (including key resources to facilitate data operations in data exploration) is proposed. The base layer of the architecture organizes data in a multi-dimensional data model to normalize the structure of the data exposed by the upper data services. The second layer, which is the resource layer, interacts with the underlying data using the data model. The resource layer provides different operations for data analysis purposes. These operations are grouped into five categories: filters, aggregators, samplers, statistic functions and packager. The third layer is the resource navigation layer, aiding in navigation through resource APIs. Through an interactive conversation between the user and data services, the resource navigation layer continuously recommends and presents interesting resources to the user by presenting links and/or suggesting parameters to make API calls.

3. Following the HATEOAS constraints of REST, a service navigation model is proposed. The approach takes advantage of REST’s properties and its related hypermedia-driven features to make resource APIs generate and navigate each other automatically based on analytical needs. To manage the service request and discover the optimal resource, resource navigation involves three-step methods that are complementary to each other.

(a) Metadata service, which provides information about the dataset schema and the available resources;

(b) A discovery tree, which helps the user to explore data based on previous data exploration queries, achieved by maintaining a binary data tree structure holding the queries executed in previous sessions;
(c) Parameter recommendation, which predicts possible interesting parameter settings based on the semantics of three types of data analytics operations (Roll up, drill down, and pan).

4. An extension of the data package as a data service mechanism is proposed that contains, publishes and shares data processing scripts so that users can customize and reuse the data exploration process. In addition, the data package is extended to provide context information for analysts, including data privacy and data provenance for analysts.

A prototype of an API framework following the DEaaS architecture was developed. The evaluation was based on both case study and experimental methodology. During validation, the refinement, implementation and evaluation of the solution proceeded iteratively until the research target was achieved as expected. Evaluation through case studies showed that this approach can enhance the interoperability and discoverability of data services and the reuse of data exploration processes. Experimental results showed that DEaaS outperformed traditional data services in terms of discovering services for data exploration.

There are three main benefits of using DEaaS data services to realize data exploration. First, pushing computation to the data side is more efficient than pulling the entirety of the data out for computation. Second, it allows the data publishers to expose a subset of data features for analytics purposes to protect their data privacy and commercial interests rather than open up the whole dataset. Finally, the data package enables data consumers to share the results of their explorations, and thus provides the benefits of reusability, flexibility and customizability.

DEaaS also facilitates the human-in-the-loop processing needed for interactive data exploration and brings domain knowledge into data service design. Moreover, by automatically recommending data queries and streamlining the data exploration,
performance in exploratory analysis is improved.

6.2 Limitation

Through the case studies and experiments, several limitations of DEaaS design were identified:

- In terms of the implementation, although the DEaaS prototype has covered most of data operations, it is still not comprehensive enough to enable analysts to perform all kinds of data exploration compared to e.g. Python or R libraries. However, the DEaaS API library can be extended on-demand in the future.

- The discovery tree of DEaaS provides a method for analysts to navigate the most popular exploration process. However, the discovery tree is not an optimal algorithm for classifying the user groups and satisfying the user-specific requirements. This could be a promising and interesting research issue for the future work.

- The discovery tree of DEaaS relies heavily on the query history. Therefore, initial service users cannot acquire help from it during the data exploration process when the tree is empty.

- The parameter recommendation of DEaaS strongly depends on the data analytics semantics, which are tightly bound to the underlying multidimensional data model. The semantics derived from OLAP operations in the approach cannot be applied for other data models.

In terms of the methodology, the evaluation method used in this research also has several limitations: In terms of the methodology, the evaluation method used
in this research also has several limitations:

- Evaluation using a case study methodology cannot provide conclusions with statistical significance; rather, many different kinds of evidence, figures, statements, and documents need to be linked together to support a strong and relevant conclusion [61]. Accordingly, in the case studies conducted in this research, a comparative evaluation of the DEaaS approach against the DaaS (Data as a Service) approach has been provided to allow for more precise conclusions to be drawn.

- In evaluating the research, the author took on the roles of both the observer specifying the case study plan and the participant involved in the implementation. To avoid the potential bias associated with this process, data were drawn from both synthetic and real-life data sources rather than subjective human perceptions.

- It is difficult to take all possible circumstances into consideration when applying a scenario-based evaluation method, and the reader may doubt whether the approach works well in scenarios other than the ones used for the evaluation. However, the scenarios adopted in this study are critical and typical in the analytics community and have been used to solve real research and industry problems [54].

- The performance metrics defined in the experimental evaluation were restricted on the accuracy of service discovery, while the others measuring the performance of query execution (e.g. response time or latency) were not considered in the system design. However, the performance metrics should be defined based on the research motivation, and the main focus of the present research was service design and recommendations.
6.3 Future Work

The planed future work includes the following:

- The 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 for realizing a REST streaming API: one involves delegating the streaming process to the REST web service, while the other involves streaming data via HTTP. We plan to study the analysis requirements for streaming data and adopt an approach that is best suited to analytics processes.

- The resource navigation consists of three main components; the meta data service and discovery tree can be applied to all resource-oriented architecture, while the parameter recommendation is mainly based on the data analytics semantics, which are tightly bound to the underlying multidimensional data model. However, we deem that this approach can also be applied to other data types and models (e.g. semi-structured data or graph data models) as long as their analytics semantics are analyzed and defined in the protocol. Accordingly, we will extend this approach to accommodate other data models in our future work.
Bibliography


discovery made efficient. In Web Services (ICWS), 2015 IEEE International

[31] Roy T Fielding. Architectural styles and the design of network-based software

[32] Felix Gessert, Steffen Friedrich, Wolfram Wingerath, Michael Schaarschmidt,
and Norbert Ritter. Towards a scalable and unified rest api for cloud data

[33] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query
recommendations for olap discovery driven analysis. In Proceedings of the
ACM twelfth international workshop on Data warehousing and OLAP, pages

[34] Till Haselmann, Gunnar Thies, and Gottfried Vossen. Looking into a rest-
based universal api for database-as-a-service systems. In Commerce and Enter-
pise Computing (CEC), 2010 IEEE 12th Conference on, pages 17–24. IEEE,
2010.

[35] Florian Haupt, Dimka Karastoyanova, Frank Leymann, and Benjamin
Schroth. A model-driven approach for rest compliant services. In Web Services

[36] Florian Haupt, Frank Leymann, and Cesare Pautasso. A conversation based
approach for modeling rest apis. In Software Architecture (WICSA), 2015 12th

[37] Joseph M Hellerstein, Ron Avnur, Andy Chou, Christian Hidber, Chris Olston,


[53] Marios Meimaris, George Papastefanatos, Panos Vassiliadis, and Ioannis
Anagnostopoulos. Computational methods and optimizations for containment

An approach to sparse, fine-grained od estimation. In *94th Annual Meeting of
the Transportation Research Board (TRB)*, 2015.

[55] Liam O’Brien, Paulo Merson, and Len Bass. Quality attributes for service-
oriented architectures. In *Proceedings of the international Workshop on Sys-
tems Development in SOA Environments*, page 3. IEEE Computer Society,
2007.

[56] 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

[57] Kai Petersen and Cigdem Gencel. Worldviews, research methods, and their
relationship to validity in empirical software engineering research. In *Software
Measurement and the 2013 Eighth International Conference on Software Pro-
cess and Product Measurement (IWSM-MENSURA)*, 2013 Joint Conference

[58] Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs:

[59] Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and
Stefan Schlobach. Linked data-as-a-service: the semantic web redeployed.


