


An Intelligent Data Service Framework for Heterogeneous Data Sources

Fakhri Alam Khan  · Mujeeb ur Rehman · Afsheen Khalid · Muhammad Ali · Muhammad Imran · Muhammad Nawaz · Attaur Rahman

Received: 28 February 2018 / Accepted: 22 May 2018 / Published online: 16 June 2018
© Springer Science+Business Media B.V., part of Springer Nature 2018

Abstract Heterogeneous data on distributed computing sources are growing day by day. To manage the data from the distributed sources into a distinct type of application like mobile, cloud, desktop, web etc. is a challenging issue in the global information systems, particularly for cooperation and interoperability. This paper proposes a Data Service Framework, which integrates the data from distributed sources such as databases, Simple Object Access Protocol (SOAP) based web services and flat files, and performs create, read, update and delete (CRUD) operations on it through Representational State Transfer (REST) services over the Hyper Text Transfer Protocol (HTTP). The proposed data service framework also supports java database connectivity (JDBC). Detailed description of the proposed framework and experimental results are reported in this paper.

Keywords Database virtualization · Simple object access protocol (SOAP) · RESTful service · Components and connectors · Software architecture

F. A. Khan (✉) · M. ur Rehman · A. Khalid · M. Ali · M. Nawaz · A. Rahman
Center for Excellence in IT, Institute of Management Sciences, Hayatabad Peshawar, Peshawar KPK 44000, Pakistan
e-mail: fakhri.alam@imsciences.edu.pk

M. Imran
IBMS, The University of Agriculture, Peshawar, Pakistan

1 Introduction

In the present day software era, much emphasis is put on the need of effective communication between cross-platform applications. Information system can now no longer operate as independent unit without putting an adverse impact on effectiveness of an enterprise due to a complex operating environment and constantly changing behavior of information systems with the passage of time. Continuous software evolution needs a flexible and scalable integration platform, which minimizes the efforts of human interaction for adaptation and maintenance [1]. A large fraction of users want to keep their data in a well-structured format, whether it is on a personal computer or network server. It is always required to use standardized application programming interface (APIs) to manage the data. If data is in relational format, then structural query language (SQL) provides the set of operations for querying data, but unfortunately the main issue is that all data is not relational. Even if the data exposed to the world is in relational format, it is not often possible in most of the cases to perform SQL operations on it over the Internet.

Nowadays, the modern applications are generally built by using two different technologies namely the object oriented programming and relational databases. Object oriented programming is used for business logic implementation in application and relational databases are used for data storage purposes. Object-oriented programming provides reusability, robustness

and maintainability for implementing complex systems. Relational databases are repositories which provide data persistency, consistency and integrity. ORM (object-relational mapping) is a bridge between the above mentioned two technologies that allows applications to access relational data in an object oriented way [2].

Heterogeneous data on the distributed sources are growing day by day. To manage the data from distributed sources in dissimilar types of applications like mobile, cloud, desktop, web etc. is acting as a stimulus for cooperation and interoperability within the global information systems [18, 19]. Also providing integrated access to multiple heterogeneous data sources including databases, flat files and web services is a challenging issue in global information systems. In this context, two fundamental problems arise. Firstly, how to integrate different sources in a single framework? Secondly, how to perform CRUD (create, read, update and delete) operations on the heterogeneous data sources from different types of applications such as web based, desktop, mobile and cloud? The current available frameworks such as REST (Representational state transfer) and SOAP (Simple Object Access Protocol) over the HTTP (Hyper Text Transfer Protocol) do not facilitate this task. Users, in most of the cases, are just unable to perform CRUD functionality using RESTful services on heterogeneous data sources.

In this paper, we propose a framework called Data Service Framework (DSF) that provides integration of heterogeneous data sources such as databases, SOAP-based web services and flat files through virtualization and REST-based protocol for CRUD-Style operations (Create, Read, Update and Delete) against all those resources which are exposed as data service. Data service framework provides services for creating and executing bi-directional data services. Through abstraction and federation, data is accessed and integrated in real-time across the distributed data sources without copying or moving data from its physical sources. In short, data service framework is the web-based equivalent of Open Database Connectivity, Object Linking and Embedding, Database, Active Data Objects for .NET and JavaDatabaseConnectivity. Major contributions of our research work are:

- A novel data service framework, which provides the facilities of heterogeneous data sources integrations.

- Our proposed framework provides services for engendering and execution of bi-directional data services.
- Real-time access and integration of data access; distributed data sources, without copying or moving data.
- Our proposed data service framework has support for Oracle, MySQL database, flat files and SOAP-based services.

Rest of the research paper is organised as follows: In Section 2, state of the art research work is explained. In Section 3 the problem under discussion is elaborated. The proposed framework is explained in Section 4. Data Service Framework and the experimental results are discussed in Sections 5 and 6 respectively. The paper is concluded using Discussion and Conclusion Sections 7 and 8.

2 Literature Review

Wada [3] proposes a database virtualization technique that uses ubiquitous databases available on the Internet by reorganizing and merging all the databases into a single database in order to apply data mining techniques on the wholesome data. This technique helps reduce the user workload such as data collection and cleansing. The proposed technique consists of the following four layers:

- First layer examines XML schema and proposes a database virtualization in such a way to represent ubiquitous databases as a relational databases and access XML and object oriented databases as a single database [13–16].
- Second layer corresponds to virtualization of ubiquitous databases that describe the ubiquitous database schema using XM.
- Next layer describes the common schema generation and query language techniques for use in the virtualized environment in ubiquitous databases.
- The final layer describes troubleshooting and recovery techniques in ubiquitous virtualized database environment.

Zhang [4] proposes a general-purpose conceptual data model called Structural Engineering Experimental Data Management Framework (SEEDMF). The model contains core data concepts involved in the

structural engineering experiments. This model proposes integration of heterogeneous data in SEE-Gird by using grid technology and also provides a solution of sharing these heterogeneous experimental data in real time. Structural engineering experiments always generates huge amount of data which can be divided into two types: experimental management information, and experimental results information. Structural Engineering Experimental Data Management Framework use GSI authentication and GridFTP is used for data transportation. Experimental management information is stored in relational database and experimental results are stored in text files; both the relational database and result files can be accessed by the corresponding services. Data access portal is responsible for interaction with business logic between data management system and the user and file transfer portal is used for data transfer and data conversion from GridFTP to HTTPS.

Three different layers of services are provided by the cloud: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Considering IaaS clouds, the major commercial cloud providers are supplying on demand virtual machines to their clients. Cloud computing brings the flexible use of resource virtually on demand. Cloud computing virtualization model enables the ability of migrating resource irrespective of the original physical infrastructure. Since open source tools are growing rapidly, so small companies build their own IaaS cloud with the use of internal organizational computing resources. The purpose of such cloud is not only to provide and sell computing resources over the Internet, but to facilitate the clients with a flexible private infrastructure to run service workloads within their administrative domains. In this context, REST-based interfaces provide the integration of third party systems with cloud enabled virtual environment. Celesti [5] developed a REST-based (Representational State Transfer) web service administration tool compatible with the Open Cloud Computing Interface. These REST-based interfaces provide the integration of third party systems with Cloud-Enabled Virtual Environment (CLEVER). The CLEVER is an open source cloud Infrastructure as a Service (IaaS) middleware allowing the allocation and management of virtual machines. All the entities within the cluster communicate with each other through Extensible Messaging and Presence Protocol (XMPP). Internal

communication between sub-components of the same module can be done through inter process communication (IPC). External communication between two different hosts can be done through XMPP, and all the hosts in the CLEVER domain must be connected to the XMPP for external communication.

Niknam [6] develop a semantic based cost estimating application for construction projects that access distributed data sources over the internet and significantly reduces human involvement in cost estimating activities. The application is based on ontologies 1) a building information model (BIM) knowledge base, (2) an estimating assembly and work item knowledge base, and (3) suppliers' Semantic Web Services. Chaudhari [7] presents a hybrid benchmark based on Transaction Processing Performance Council (TPC-H) where heterogeneous data source such as relational & XML data sources and redesign of query language LINQ is used to assess the data heterogeneous data sources.

Szepielak [8] states that web services perform all types of operations in SOA-based integration approach. However, these services are often impossible to utilize without prior adaptation to their interfaces. To overcome this problem, resource-centric approach is used where interoperability problem is tackled by providing a fixed functional interface, but this can limit the flexibility of operation definition. REST-based architecture is generally used for resource-centric approach because of its simplicity and mapped in a natural way. XML document is used for uniform resource representation and HTTP methods are mapped on create (POST), retrieve (GET), update (PUT) and delete (DELETE) operations. REST-based service provides the create, read, update and delete (CRUD) functionality for the resource and complete set of services are accessed from the resource service pool, which provide the uniform access to data stored in the underlying information system. Web service registry is the main component of a dynamically integrated environment. The registry uses the conceptual model provided in the ontology to build a service directory. It also extends the conceptual model representation of the ontology to include information about the providers and operations.

Atay [9] proposes extensible markup language (XML) to relation data mapping technique in order to store and query XML documents using relation

database managements system (RDBMS). XML to relation data mapping is of three types: Schema mapping, Data mapping and Query mapping. In schema mapping, generic database schema is generated from XML document type definition (DTD) schema to store XML documents. In data mapping the XML documents are divided into relational tuples and are inserted into relation database. In query mapping, XML query is converted into relational database and query result are returned to user as XML document. For the sake of simplification, the DTD is divided in canonical form to convert it into the database schema. The occurrence operators in DTD are classified into two groups on the basis of their relationship between parent and child. One to one relationship is represented by operator {'?', ','} and one to many relation is represented by operator {'+', '*'} . Through this approach, the complex DTD can easily be converted into the relational database schema with the help of identifying the operators group in DTD.

Pautasso [10] discusses how composition can be applied on service oriented architectures for building Restful service in order to make their use more effective. Composition is one of important principle of service oriented computing [11, 12]. It flourishes the exiting services by assembling them into multiple applications in innovative and amazing ways. This principle is not defined in the REST architecture. Instead, the REST architecture focuses on REST architectural elements (user agents, gateways, proxies and origin servers)which are meant to be combined together to build a scalable system and it helps access the published resource on a single server by a large number of clients. It is pertinent to mention that each published resource is accessed over the HTTP protocol.

Luo [17] propose prototype system for integration of heterogeneous medical data resources based on Grid technology. The proposed prototype model implemented and tested in a simulated information environment. In this environment they integrate heterogeneous collection of machines with installation of different medical databases. All machines are connected with each other through a 100-m local area network (LAN). The results shows that they still need to improve the efficiency, dynamic metadata modeling and Security.

Alghamdi [18] optimize the structural and constant part of XML queries by introducing the method of indexing and processing XML data based on the

concept of objects that is formed from the semantic connectivity between XML data nodes. This method performs object-based data partitioning, which goal is leveraging notion of frequently-accessed data subsets and putting these subsets together into adjacent partitions. Structural and Content indexing, which use an object-based connection to construct indices and query processing to produce the final results in optimal time.

Most of work mentioned above mainly initiative in a distributed environment with is looking at homogeneous data sources integration and access of data management only [20, 21]. The main objective of our work is to make a framework which will integrate the heterogeneous data source and access of those data source from a single framework without copying data locally and data manipulation operations from distinct types of applications such as mobile, desktop and web-based applications, so that enterprize solution can benefit in terms of saving time and cost from the effort being put in.

3 Problem Statement

Integrating heterogeneous data on distributed data sources such as databases, SOAP-based services and flat files, and accessing this distributed data sources in an information system pose two key challenges. Firstly, how to integrate these different data sources into a single framework; and secondly, how to perform the data manipulation operations from distinct types of applications such as mobile, desktop and web-based applications. In this paper, we address the above-mentioned two main problems. We propose a Data Service Framework which integrates distributed data sources such as MySQL or Oracle databases, SOAP based services and flat files (text files either comma or tab delimited) into a single framework, and provides a unified data manipulation access from different types of applications such as mobile, cloud, desktop and web-based applications. This framework can help the enterprize solutions in terms of saving development cost and time. The following advantages are likely to be obtained from the data service framework.

- Enterprize services for heterogeneous data access.
- Data integration or virtualization of data from heterogeneous models and sources.

- Data interoperability.
- Data access on the web.
- User identity, authentication and authorization.

4 Proposed Framework

In view of the problem statement mentioned in the previous section, we propose a Data Service Framework (Fig. 1) which supports integration of heterogeneous data sources such as databases (Oracle and MySQL, SQL Server), Simple Object Access Protocol (SOAP)-based web services and flat files. The proposed data service framework consists of three layers. The first layer caters for the heterogeneous data source integration and chalks out the scheme for extracting and manipulate the data without data copying or moving it physically. The second layer explains the object relation model creation on virtual database layer. In the third layer, we define methodologies to expose object relation model to the world through RESTful services.

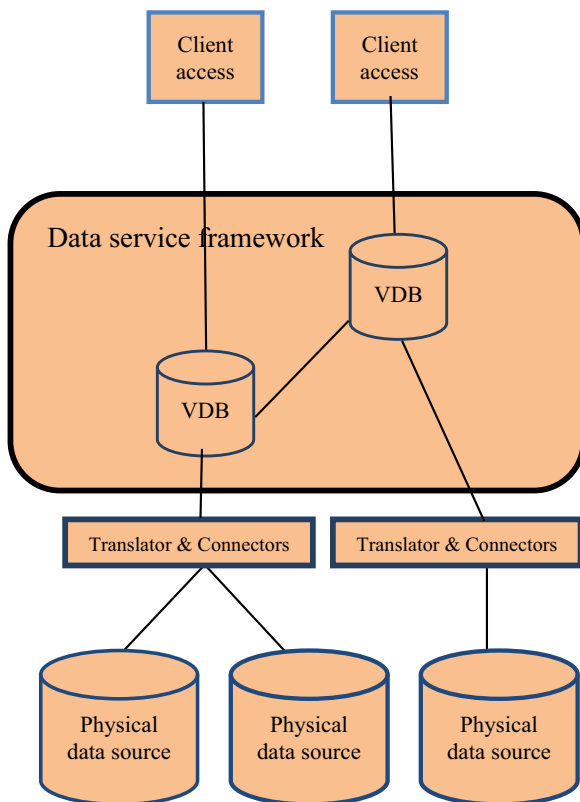


Fig. 1 The proposed data services framework

The first layer of the framework explains the data integration mechanism. The data integration is done through a virtual database in such a way that data cannot be copied or moved from the physical location. Virtual databases (VDBs) are virtualized data sources that expose a composite schema for one or more data sources. VDBs can be hierarchical in the sense that their underlying data sources could themselves be VDBs. VDBs are purely “virtual” data sources and do not store any data; they simply expose the composite view of the data from the underlying data sources. When a query is issued against a virtual database, then data service framework uses the query engine which receives the query, resolves it into sub-queries for the underlying physical data sources and splits it into multiple fine grained queries. There are different types of query engines such as Quest, Teiid, and SQL Query Engine for Big Data and Presto. In our proposed data service framework, Teiid query engine is used to account for the required transformations and merges (for federated VDBs) to build the final result set. When a logical schema is deployed in Teiid run time query engine, then it behaves like a relation database. Teiid is a transitionally responsive system and data is extracted and manipulated in a secure way. Its connectors provide the data access mechanism to physical data source such as relation database, flat files and SOAP services. Another important feature of connector is to map the underlying physical data source model in relational entries such as tables, views and procedures.

Teiid query engine uses two separate connectors and translator to execute query on a physical data source. When SQL query is submitted, query engine uses the logical schema information to parse, optimize, validate and split the query into multiple physical source specific queries. The second connector executes queries on the physical data sources and extracts the data and processes it into a final result set. But the issue is that the physical data sources can have different command syntax, so for this purpose the command execution connector needs to have a translator. Translator provides an abstraction layer between physical data sources and query engine. Translator executes the command using specific physical data source adaptor. It also converts the result set data which is extracted from the physical data sources into the query engine format. Translator and resource adaptor are configured in the source model layer. Physical

data source connectivity is done through resource adaptor. Resource adaptor can be of different kinds like relation database, SOAP based service, text files and custom resource adaptors. The result set fetched by the resource adaptor is fed to the translator. In case the transaction is performed on distributed data sources, then distributed XA (eXtended Architecture) transaction is used through JCA connector.

Every VDB has one or more logical schemas and these schemas are exposed as “data models”. There are two types of data models in a VDB: view models and source models. View models represent the schema defined by the VDB, while source models represent the schema of the underlying data sources that constitute the VDB. The Data Manager determines the view model (schema) based on business needs and the knowledge of the underlying source models, and publicizes the name of the view model for clients to consume. Source models are typically not advertised and are therefore not visible to client applications. VDBs can be hierarchical; view model of a VDB will become a source model for subsequent higher VDBs. If the Data Manager makes changes to a published view model, then a new version of the model is created. Hence, it is important to know not only the data model but the version of the data model that the VDB is composed of. There are different types of VDBs.

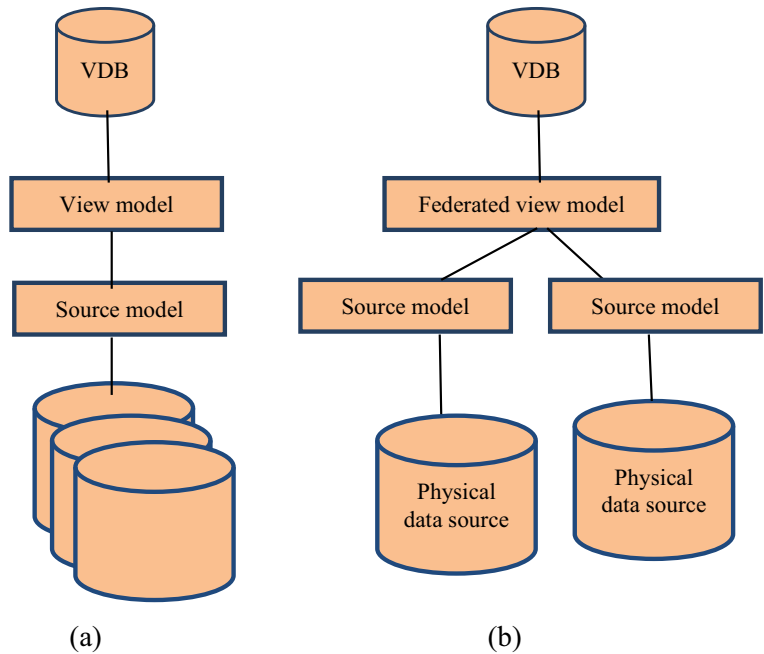
Any VDB created outside of the Teiid design tool is called “Dynamic VDB”. The ones created by the Teiid designer tool are called “Standard VDB” or “Designer VDB”.

Though a VDB is defined in a structured XML file, but can be packaged in multiple ways. If the VDB definition is straightforward, then the XML file can be directly deployed. If the VDB definition involves the Teiid designer tool, the VDB is packaged as a “.vdb” archive file that contains the VDB definition in an embedded XML file. It is not important to differentiate as how the VDBs were created, but it is important to identify whether the VDBs expose data from a single source model or multiple source models. A brief account of both the options (Fig. 2) are defined as follows:

a. VDBs that expose data from a single source model:

These VDBs expose view models that represent data from a single source. In fact, data may be exposed from multiple data sources, but they all share the same source model. If more than one underlying data sources are exposed, the Data Server automatically adds a virtual column named “MULTI_DATA_SOURCE_COLUMN” to all the entities to uniquely identify the source of data for each row.

Fig. 2 a Single Source View Model, b Multiple Source View model



b. VDBs that expose data from multiple source models:

These VDBs expose view models that represent data from multiple source. Data is federated from multiple underlying data sources that do not share the same source model. There is typically no direct mapping from source model to the view model, and a “join” is involved when creating the view model from multiple source models. Within each VDB, the underlying data sources are identified via a unique name, and this unique name is used to extract data from specific data sources.

Another type of VDB called “Admin VDB” is used to fetch information about all the models and physical data sources that are hooked with the data service framework. This VDB is useful for client applications for discovering the VBs deployed in data service framework before actually connecting to a particular VDBs.

In the second layer of data service framework, we use Object Relational Mapping (ORM) in virtual databases because the classical databases access approach falls short of the merits of object oriented modal. Nevertheless, the key issue of data representation in object oriented paradigm still remains. Such an issue is well addressed by ORM frameworks such as Hibernate. Therefore, a modal class is needed for all of the virtual database views where each VDBs is represented in a class, so that there is one-to-one mapping for each individual column and data types of the virtual database. Keeping data in this representation carries several advantages like session maintenance, and sometimes the data retrieval becomes easier. This data representation layer adds an additional level of

buffering to avoid overloading databases with write access load. Once a session commits, the write operation is sent to the query engine. This not only optimizes the database write operation but also make it possible for each individual application program call to interact with databases in a non-blocking way. Similarly, rollback operation does not cost too much as it is generally conceived in a way that it just discards local object memory instead of invoking disk operations.

The last layer of data service framework explains the methodology of data access. VDBs are accessed as a typical relational SQL data sources even if the underlying physical data source may not be relational or structured. Programmatically, they can be accessed either using the REST-based Data Server Web Service or using a compliant JDBC or ODBC driver. The detailed architectural design of the data services framework is illustrated in Fig. 3.

Client machines can access data in the data service framework in one of two possible ways:- REST-based web service access or JDBC/ODBC Driver interface access. In REST-based web service access, the Data Server web service exposes VDBs as “resources” and allows web clients to query these resources and their entities using the OData(Open Data) protocol. OData is a standardized HTTP-based protocol to expose, query and interact with the data sources. It provides full metadata of the data source (with a \$metadata query). HTTP payload conforms to the ATOM and JSON formats. It is worth mentioning that application programs consume data exposed using the OData protocol [10]. The Data Server web service exposes the following resources:

Data Models: <entryPoint> (e.g.http://<HOST>:<PORT> /DSFServer/dsl.svc)
Data Model Version:<entryPoint>/<modelName>
VDBs:<entryPoint>/<modelName>/<modelVersion>/
OData Service Document (List of Entities for a VDB):
 <entryPoint>/<modelName>/<modelVersion>/<CompositeNameOfVirtualDataSource>/
Entity Details:<entryPoint>/<model>/<modelVersion>/<CompositeNameOfVirtualDataSource>/<EntityName>

The <CompositeNameOfVirtualDataSource> is an important concept to understand because it varies based on the view model of the VDB, and a VDB exposes data composed from one or more data sources. Depending what type of data in exposed by the VDB, the <CompositeNameOfVirtualDataSource>

is either just the VDB name, or a composite string made up of two parts separated by a hyphen ('-'); the VDB name and the underlying data source name. While the VDB name is obvious, the underlying data source name varies based on how the VDB is constructed. In this regard, single

source-model VDB and multiple source-model VDB are considered.

While for JDBC/ODBC Driver interface access, the Data Server exposes the VDBs as SQL based relational databases that can be accessed via a dedicated ODBC or JDBC driver.

4.1 Deployment

Data service framework needs to be deployed/hosted on web server. For this purpose we use JBoss Application Server which is an open source Java EE-based application server.

4.2 Accessing Data Using a Browser

Data in our framework is accessed using OData. Although this kind of access is read-only, it is very useful to verify data model. The responses from Data service framework are Atom Feeds. The set of procedure to access the data include:

- i. Open a browser and navigate the following address to get a list of the models:
<http://<HOST>:<PORT>/DSFServer/dsl.svc>.
- ii. DSF will be asked for the login and password information.

iii. Once the user is validated, a list of data models is displayed in the form of data service URL.

iv. We can access the data models by appending the model name to the base URL.

For example, <http://<HOST>:<PORT>/DSFServer/dsl.svc/ExampleModel/>

v. We can access the data sources deployed under the model and version combination by appending either the model version or user version to the URL.

Forexample, <http://<HOST>:<PORT>/DSFServer/dsl.svc/ExampleModel/>

The entity model is linked to model and version combination, whereas the data sources list different physical/virtual sources that hold the actual data for that model. We can get the metadata for the entity model by using the \$metadata resource on the URL. The metadata provides the following information:

- i. All entities, relationships, complex types and functions defined in the entity model. This basically provides the snapshot of the entity

model that we can use to retrieve/update data for the individual entities.

ii. Annotation defined by the OData specifications and custom annotations defined by data service framework.

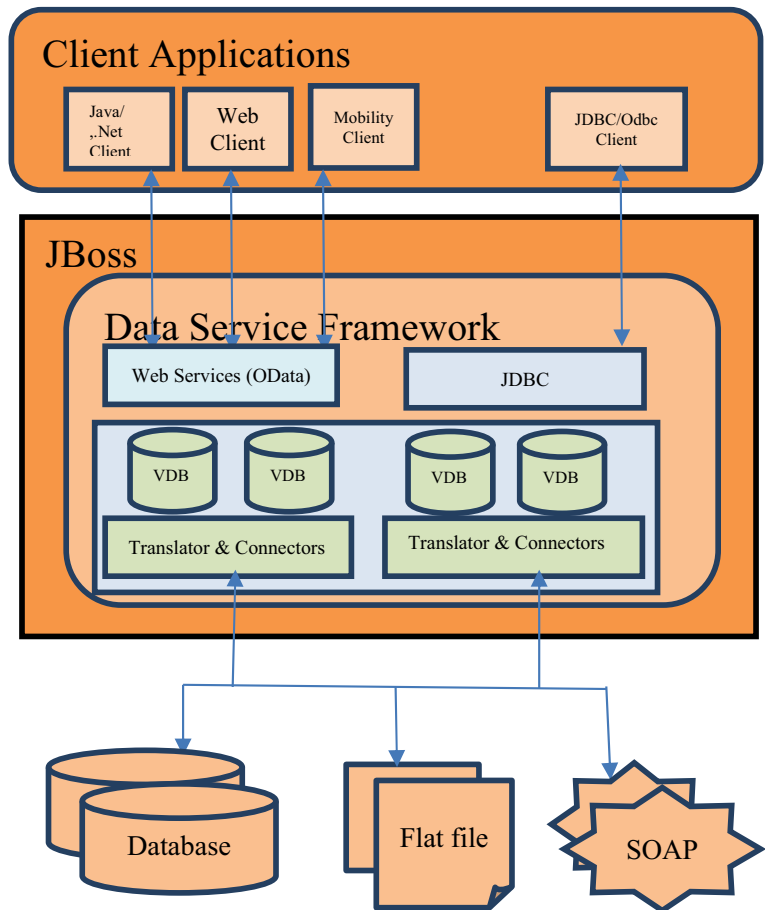
iii. The OData version supported by the entity model

Some characters have special meanings when they are used in a URL. In queries executed at the URL, special characters are specified as %xx, where xx is the hexadecimal value of the character. If user wants to use the special characters in the URLs as query parameters, then they need to be encoded with their respective hexadecimal value.

4.3 Deleting Data

Regardless of how can we access data, data service framework does not support cascade delete of entities. We must delete all child records in the hierarchy first before deleting the parent record. If we try to delete parent without deleting its children first, then delete

Fig. 3 Architectural design of the data services framework



will fail and the corresponding error message can be seen in the server logs.

4.4 Versioning RESTful Services

The Data Server RESTful resources consist of URIs, web resources, web content representation, web content format and the HTTP methods for each web resource. Versioning becomes handy when isolating changes to Data Server RESTful resources from existing Data Server clients. Versioning a RESTful web service involves versioning resources with new URIs. The idea is to treat each version of the resource as a different representation so that clients can negotiate for a given version by submitting the model version number. If the server supports that version it will return a representation of that version. This gives the Data Server clients the option to work with and switch among multiple versions of the resource.

5 Data Service Framework Usage

Real applications often access more than one data source. Many of the enterprise built their own frameworks to handle integrating multiple sources, and have realized the difficulty of doing that in a generic manner that performs and scales well under real use conditions. The Data Service Framework (DSF) provides applications with a common data access to distant heterogeneous data sources. It enables access to data via services instead of individual development kits for different databases, flat files and SOAP services access. The Data Service Framework (DSF) also provides tools for developers and consultants to create connections to additional data sources and expose the data as services. Custom and hand-coded logic frameworks will not be required while using DSF, and use a dedicated query component for all data access needs of enterprise application. Due to the use of DSF


```

← C http://dsf.com/5433/Cloud-dest-pr/cloud2/1/file-vids/mimg/rabbitmq
2018-02-16 11:22:14.689 [info] <0.33.0> Application lager started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:14.696 [info] <0.5.0> Log file opened with Lager
2018-02-16 11:22:28.556 [info] <0.33.0> Application jsx started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.588 [info] <0.33.0> Application os_mon started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.588 [info] <0.33.0> Application recon started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.589 [info] <0.33.0> Application inets started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.639 [info] <0.33.0> Application mnesia started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.639 [info] <0.33.0> Application xmerl started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application asn1 started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application crypto started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application public_key started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application cowlib started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application ssl started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application ranch started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application ranch_proxy_protocol started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.640 [info] <0.33.0> Application rabbit_common started on node 'rabbit@PK-LTP-MujeebRehman'
2018-02-16 11:22:28.650 [info] <0.213.0>
Starting RabbitMQ 3.7.2 on Erlang 20.2
Copyright (C) 2007-2017 Pivotal Software, Inc.
Licensed under the MPL. See http://www.rabbitmq.com/
2018-02-16 11:22:28.655 [info] <0.213.0>
node       : rabbit@PK-LTP-MujeebRehman
home dir   : C:\WINDOWS\system32\config\systemprofile
config file(s) : c:/Users/PK/AppData/Roaming/RabbitMQ/advanced.config
              : c:/Users/PK/AppData/Roaming/RabbitMQ/rabbitmq.conf
cookie hash : +J5rrrWtztVdwXspTwYoVbQ==
log(s)     : C:/Users/PK/AppData/Roaming/RabbitMQ/log/RABBIT~1.LOG
              : C:/Users/PK/AppData/Roaming/RabbitMQ/log/rabbit@PK-LTP-MujeebRehman_upgrade.log
database dir : c:/Users/PK/AppData/Roaming/RabbitMQ/db/RABBIT~1
    
```

Fig. 5 Snapshot of the flat file access

implemented DSF in a simulated environment and tested it. Consequently, following issues need to be addressed for using DSF in enterprise applications:

High Availability For large scale enterprise applications, we need a high availability. Currently, there is no mechanism in data service framework if primary physical source goes down and it cannot be up from secondary/slave node. For this purpose, we need to

change the address of physical source in data service framework.

Security Data security is a crucial requirement which mainly depends upon user authentication and authorization, and data transport layer mechanisms. Data service framework saves user credentials and access rights in an encrypted configuration file. Moreover, current authentication and authorization mechanisms

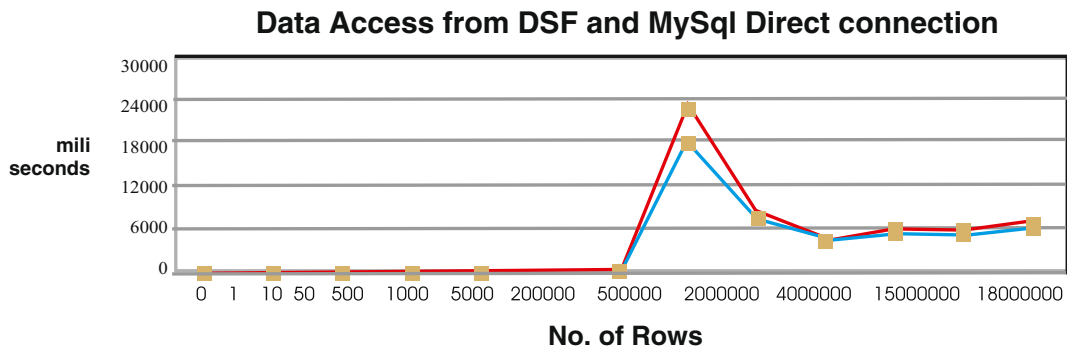


Fig. 6 Snapshot exhibiting the comparison graph

cannot address all the issues that arise in enterprise applications. Resultantly, new authentication and authorization as well as data transport layer mechanisms need to be developed to achieve the special security requirements of integration of heterogeneous data source in DSF.

8 Conclusion and Future Work

Heterogeneous data extraction and manipulation from distributed sources including databases, files and web services in distinct type of application like mobile, cloud, desktop, web etc. is a challenging issue in global information systems for cooperation and interoperability. In this paper we proposed a framework called data service framework, which provides the integration of heterogeneous data sources and services for engendering and executing bi-directional data services. Through abstraction and federation, data is accessed and integrated in real-time across distributed data sources without copying or moving data from its system of record. This framework helps the enterprise solutions in terms of saving development time and cost. It also helps different types of application such as mobile, cloud, desktop and web-based to access the heterogeneous data from distributed source without developing the business logic layer again and again. Thus, the perceived simplicity of DSF now can be understood from a quantitative perspective – choosing DSF removes the need to creating different application for different data sources as well as data management. The DSF as shown in experimental result will access data from heterogeneous data sources, in terms of performance we have verified the performance on MySQL data source to run the same query on MySQL using DSF from client machine and also created a direct connection with MySQL to access the virtual machines record and we found the difference of 6 seconds while accessing 200 hundred thousand rows. But while accessing the lower numbers of rows the difference goes to milliseconds.

The proposed framework is currently supported for Oracle, MySQL databases, flat file (text) and SOAP-based services. As a future work, it can be extended for other databases such as SQL server, SQLite and DB2. Also Extensible Messaging and Presence Protocol (XMPP) is required to be integrated at translator

level in query engine to handle the situation where physical resource goes down and we want to power it up from the slave system automatically.

References

1. Russell, C.: Bridging the object-relational divide. *ACM* (2008)
2. Vinoski, S.: Verivue: RESTful web services development. *IEEE Internet Computing* 5(3) (2008)
3. Wada, Y., Watanabe, Y., Syoubu, K., Miida, H., Sawamoto, J.: Virtual database technology for distributed database in ubiquitous computing environment. *Amer. J. Database Theory Appl.* 1(2), 13–25 (2012). <https://doi.org/10.5923/j.database.20120102.02>
4. Zhang, X., Di, R.H., Weng, Z.J.: Data management in structural engineering experiment grid. In: *The 5th annual China grid conference* (2010)
5. Celesti, A., Tusa, F., Villari, M., Puliapito, A.: Integration of CLEVER clouds with third party software systems through a REST web service interface. In: *2012 IEEE symposium on computers and communications (ISCC)* (2012)
6. Niknamand, M., Karshenas, S.: Integrating distributed sources of information for construction cost estimating using semantic web and semantic web service technologies. In: *Automation in Construction* (2015)
7. Chaudhari, M.B., Dietrich, S.W., Ortiz, J., Pearson, S.: Towards a hybrid relational and XML benchmark for loosely-coupled distributed data sources. *Journal of Systems and Software* (2015)
8. Szepielak, D.: Rest-based service oriented architecture for dynamically integrated information systems. *J. Digit Imaging.* 22(6), 559 (2009). <https://doi.org/10.1007/s10278-009-9244-2>
9. Atay, M., Chebotko, A., Liu, D., Lu, S., Fotouhi, F.: Efficient schema-based XML-to-relational data mapping. *Inf. Syst.* 32, 458–476 (2009)
10. Pautasso, C.: On composing restful services. *Cascon* (2012)
11. Gounaris, A., Comito, C., Sakellariou, R., Talia D.: A service-oriented system to support data integration on data grids
12. Strimbei, C.: Smart data web services. *Informatica Economică* 16(4) (2012)
13. Wada, Y., Watanabe, Y., Sawamoto, J., Katoh, T.: Database virtualization technology in ubiquitous computing. In: *Proceedings of the 6th innovations in information technology (Innovations'09)*, pp. 170–174 (2009)
14. Wada, Y., Watanabe, Y., Syoubu, K., Sawamoto, J., Katoh, T.: Virtualization technology for ubiquitous databases. In: *Proceedings of the 4th workshop on engineering complex distributed systems (ECDS 2010)*, pp. 555–560 (2010)
15. Wada, Y., Watanabe, Y., Syoubu, K., Sawamoto, J., Katoh, T.: Virtual database technology for distributed database. In: *Proceedings of the IEEE 24th international conference on advanced information networking and applications workshops (FINA2010)*, pp. 214–219 (2010)

16. Wada, Y., Watanabe, Y., Syoubu, K., Miida, H., Sawamoto, J., Katoh, T.: Technology for multi-database virtualization in a ubiquitous computing environment. In: International workshop on informatics (IWIN2010), pp. 89–96 (2010)
17. Luo, Y., Jiang, L., Zhuang, T.-G.: A grid-based model for integration of distributed medical databases. *J. Digit Imaging*. **22**(6), 579–588 (2009). <https://doi.org/10.1007/s10278-007-9077-9>
18. Alghamdi, N.S., Rahayu, W., Pardede, E.: Semantic-based structural and content indexing for the efficient retrieval of queries over large XML data repositories. *Future Gen Comput. Syst.* **37**, 212–231 (2014)
19. Imran, M., Hlavacs, H., Khan, F.A., Jabeen, S., Khan, F.G., Shah, S., Alharbi, M.: Aggregated provenance and its implications in clouds. *Future Gen. Comput. Syst.* **81**, 348–358 (2018)
20. Imran, M., Hlavacs, H., Haq, I.U., Jan, B., Khan, F.A., Ahmad, A.: Provenance based data integrity checking and verification in cloud environments. *PloS ONE* **12**(5), e0177576 (2017)
21. Khan, F.A., Ahmad, A., Imran, M., Alharbi, M., ur Rehman, M., Jan, B.: Efficient dataaccess and performance improvement model for virtual data warehouse. *Sustain Cities Soc.* **35**, 232–240 (2017)