REMOTE KEY/VALUE DATABASE

Operational Concept Document

CSE 681 Software Modelling and Analysis
Fall 2015
Dr James Fawcett

Rupan Talwar
SUID: 402408828
MS Computer Engineering

Table of Contents

1 Introduction

1.1 Executive Summary

The following Operational Concept Document details the Remote Key/Value database project. It explains various features of the project including its users, architectural structure, UI Views, application activities, critical issues and applications for which it can be used in its futurity.

Key/Value database is essentially a noSQL database, and thus it supports features and independence that comes with a noSQL database in the form of schema-free architecture, horizontal scalability and capability to handle large streams of data. In this Operational Concept Document and the project following it, we explore how a non SQL database can be constructed and used.

Owing to its distributed nature, the Remote Key/Value database would find its utility in applications that require to access remote databases.

The users of such a database vary in terms of scale and their capacity for storage requirements, yet a noSQL database seems to be fit for all the below mentioned users.

Developer, be it a student developer or a professional developer, the first hands-on user to a database after or while it is being constructed will be its programmer itself.

Instructor, an instructor acts as a user here owing to the project requirement of demonstrating the design features.

Teaching Assistant, a teaching assistant acts as a supplement to an instructor and hence is also counted as a user.

Software System, a noSQL database such as a Key/Value database may find various software using it.

Institution or Organization, these include organizations that deal with large sets of data such as supply chain organizations, social media, etc.

The Remote Key/Value database is composed of several packages that make up its architecture. These are listed below but not limited to,

Communication Module, defines a package that provides a sender and a receiver blocking queue to support message passing between the client and server.

Message Parser Package, defines a class that would order the messages in a specified structure to be sent across the communication channel to the server or vice-versa.

DBElement Package, defines the generic key/value pair contents. Here the value consists of a metadata (that comprises of a name, description, timestamp and children of a key) and the actual data payload.

Query Engine Package, defines the modules that as the name suggests deals with the formation of queries.

Database Engine Package, defines the module that would hold the C# dictionary class member.

Database Factory Package, defines the module that would act as a virtual Database, for faster query processing.

XML Parser Package, as the name suggests it helps parse XML elements to store in the form of key/value pairs in the in-memory database and vice-versa. It also manages the persistence engine and helps schedule the persists.

DBExtensions Package, this package provides extension methods to support display of various types of DB elements.

Lastly, the Remote Key/Value database experiences various critical issues that have been acknowledged in this document, these critical issues if not addressed properly may impact the design and usability of the project.

Database (or Query) Performance, An important part of this project is to consider the query performance of the Clients. This performance would differ with different types of messages.

Concurrent Access to Remote Key/Value Database by Read and Write Clients, The application involves multiple Read and Write Clients to interact with the database simultaneously, there might be a contest as to who would access the database first, considering two or more Clients run on parallel threads at the same time.

Message passing using Blocking queue, situations when the there are an overwhelming number of messages in the blocking queue, more than what the Server has been designed to handle.

Concurrent Write to the same Key, situation where two Clients try to write to the same Key.

Embed application logic in Client GUI, the temptation of dumping all the code behind each dialog window control to the respective button handler. This is a malpractice and even though it might sound easier to do, it usually causes the system to slow down considerably.

Client crashes after sending a request, there can be a scenario where a client can crash after it has sent the request but not received response.

Key-Value deletions, with the deletion of Keys, comes the question of deleting its subsequent children as well or not. Deletion of the children of a particular key that is deleted, might cause various application inconsistencies and hence is an issue of concern.

Dictionary look-up, Dictionary is a constant time operation if the key of the key/value pair being searched is provided. However, it turns into a linear search through all the key/value pairs if a specific key is not provided. Thus leading to large time taken for query processing.

Version Control, version control is used to manage medium to large software systems in order to provide them with a sense or organization and consistency.

1.2 Purpose

NoSQL databases essentially consists of a wide variety of different database technologies that amalgamate in such a manner to form a noSQL database. Its purpose was to respond to the high volumes of data utilized by users, objects and products everyday with high amount of frequency in terms of data accessibility.
On the other hand, relational databases were not designed to cope with such high volumes of data in a time and resource-efficient manner, nor were they built to take advantage of cheap storage and processing power available today.

Thus defining the following points can sum up the purpose of a noSQL database such as a Key/Value database,

i. **High Data Velocity**, large amounts of data coming in fairly quickly, and through possibly different locations
ii. **Data Variety**, storage of structures, semi-structured and un-structured data
iii. **Data Volume**, Terabytes or Petabytes of data size
iv. **Data Complexity**, Data stored and managed from different locations or data centers

1.3 Requirements

The Key/Value Database project shall implement the following the functional and non-functional requirements.

1.3.1 Functional Requirements

The Key/Value database project shall implement,

i. A generic key/value in-memory database where each value consists of a metadata (that comprises of a name, description, timestamp and children of a key) and instance of a generic type.
ii. Addition, deletion and editing of key/value pairs including their metadata.
iii. Persistence of database contents to XML file.
iv. Support for query processing on various key(s) conditions.
v. Support creation of a write client that sends data to the remote server.
vi. Support creation of a read client that would read data sent from the remote server.
vii. Support for measurement of performance metrics of the messages being sent between the clients and the server.

1.3.2 Non-Functional Requirements

The Key/Value database project shall implement,

i. C# using the facilities of the .Net Framework Class Library and Visual Studio 2015.
ii. Display shall be implemented in the form of Windows Presentation Foundation (WPF).
iii. A test executive that clearly demonstrates meeting of all functional requirements.

1.4 High-Level Design

The top-level architecture of the Remote Key/Value database includes two components,

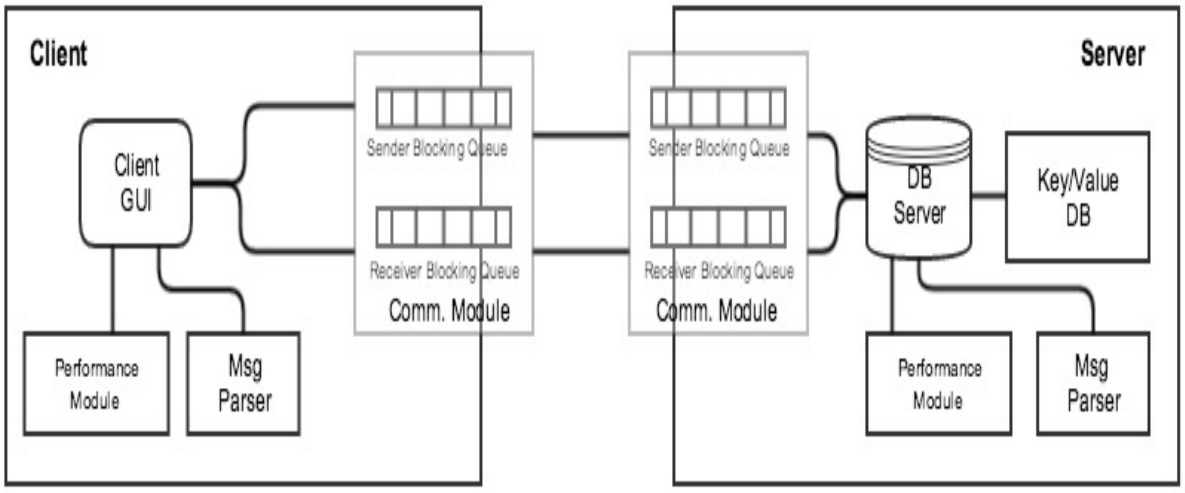

Figure 1. High-Level Architecture Diagram for Remote Key/Value Database

1.4.1 Client

The Client side of the application consists of four major components. Namely, the Graphical User Interface (GUI), Performance module, Message parser and the Communication module.

Graphical User Interface (GUI) defines the Client display. It acts as the face of the application and would be the only point-of-contact to the user itself. The client GUI and its sample design are explained in detail later in this document.

Performance Module defines the package that is responsible to measure performance metrics of the messages sent from the client to the server.

$$Performance\ Metric_{Client} = \frac{Database\ Queries\ (or\ Messages)\ received}{Per\ unit\ time}$$

Message Parser defines the module that consolidates the message to be sent to the server by the client. This consolidation helps in the transport of the message across the communication channel.

Communication Module defines the communication structure of the client. It consists of sender/receiver blocking queues to facilitate message passing.

1.4.2 Server

The Server side of the application is made up of 4 major components. Namely, the Communication module, Message parser, Performance module and the DB server that links to the rest of the Key/Value DB logic.

Communication Module defines the communication structure of the server. It consists of sender/receiver blocking queues to facilitate message passing.

Message Parser defines the module that consolidates the message to be sent back to the client by the server. This consolidation helps in the transport of the message across the communication channel.

Performance Module defines the package that is responsible to measure performance metrics of the messages sent from the server to the client.

$$Performance\ Metric_{Server} = \frac{Database\ Queries\ (or\ Messages)\ sent}{Per\ unit\ time}$$

Key/Value DB defines a broad category that encapsulates the entire database logic that helps in query processing, addition/deletion/edit of records and persistence of database into XML files.

2 Uses

The Uses focus on understanding the architecture of the product, the needs of the users of the product, the design implications that may arise with different users using this product and the requirement of the product in various environments.

The following diagram addresses the top-level context diagram of the project and mentions the wide variety of users that may use the Key/Value Database.

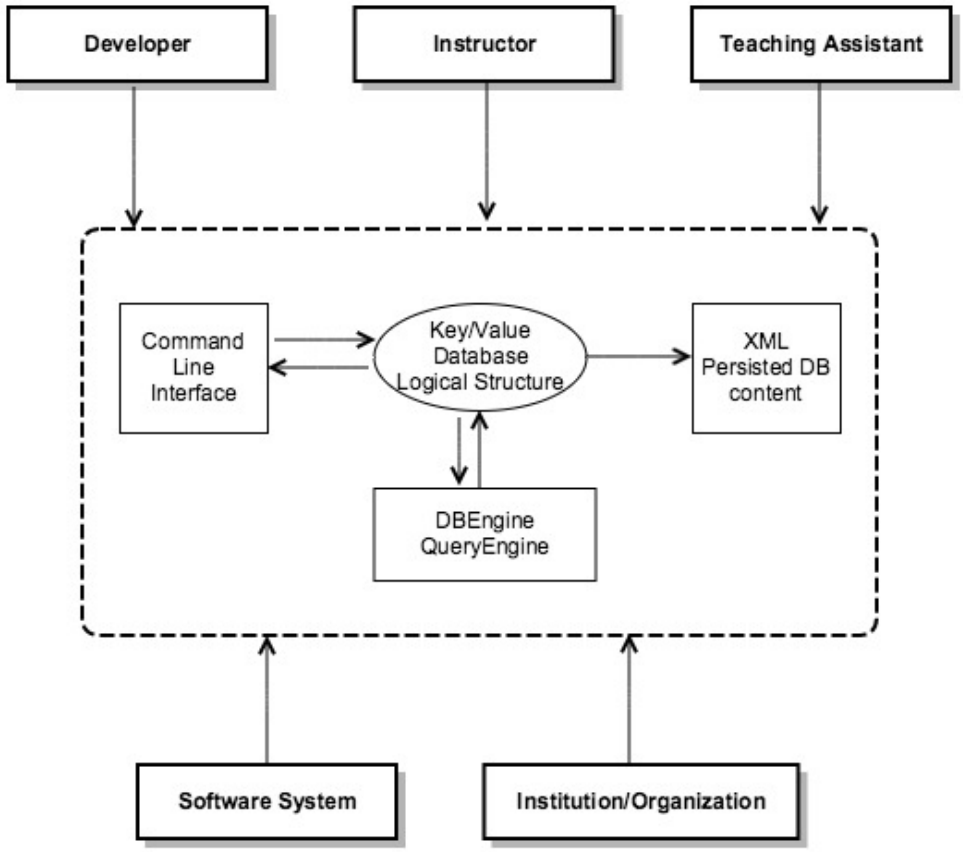

Figure 2. Context Diagram for Key/Value Database

2.1 Software Developer

A software developer, be it a student developer or a professional developer would form the primary user of a Key/Value database, since it is the same user that develops the project and thus inherently uses it.

On the other hand, if we compare the developer to a database administrator, we notice that a developer would be willing to use a noSQL daabase such as a key/value database owing to its schema-free and programmer friendly nature.

It is not necessary that a database administrator is the only end-user of such a database.

Design impact: The design implication of having a developer be a user of such a system is that the system needs to be more developer oriented, in terms of technologies used to implement varied packages in the project. The developer must be able to understand and/or tweak the package code structure if need be, for this the system must be understandable by the developer.

2.2 Instructor

An instructor forms a user to such a database owing to the project obligation of demonstrating the requirements. The instructor may use such a key/value database in order to manage real-time data about his/her class. This data may vary from in-class attendance to survey quizzes taken in the classroom.

Design impact: The design implications may vary from course to course and may also be different for each instructor. Generally, a design implication might include not being able to understand or relate the requirements to the requirement demonstration.

2.3 Teaching Assistant

A teachning assistant acts as a supplement to an instructor, and thus the needs of this user might as well mimic the needs of an instructor. This may again include management of real-time data about his/her class. This data may vary from in-class attendance to survey quizzes taken in the classroom.

Design impact: The design implications may also be same as the one stated for an instructor with minor changes. Generally, a design implication might include not being able to understand or relate the requirements to the requirement demonstration.

2.4 Software System

A software system might be a paramount user of such a key/value database. Cognizant of the fact that noSQL databases are indeed very useful for managing large amounts of data, software that deals with such large amount of data would be an active user of such a database. For example: Software such as *JBoss* by *RedHat*, open source software that supports enterprise middleware uses a noSQL database, *Cassandra*, as its back-end. *Adobe's* content management solution software uses noSQL database, *MongoDB* for scaling purposes.

Design impact: Varied software systems have varied input requirements and a key/value database might not fit into the structure of inputs required for a software systems, say a cloud based software might take inputs using a Restful API, whereas a legacy software based on mainframes might take a different input.

2.5 Institution or Organization

Companies and institutions also form users of a noSQL database such as a key/value database. Supply chain organizations, health care analytics and social media are some of the organizations that deal with large amount of data and large frequency of data are actively adopting noSQL for its various benefits. For example, *Facebook* and *Twitter*, uses noSQL database, *MongoDB,* to support real-time user analytics.

Design impact: The design implications on the database due to a wide variety of users are mainly centeres around the use of different technologies in different companies that might use such a database.

3 Views

This section provides a detailed explanation of the sample UI views that the user may face in the actual application. It consists of UI windows to suffice most of the project requirements that includes Connection window, Client Operations window, Read Client window and Write Client window.

3.1 Connection Window

The Connection window, as the name suggests, helps in provision of connection of the Clients and the Server. Since the requirements entail showcasing more than one Client, the design incorporates two Client ports to be active simultaneously.

On the other hand, there would be just one Database Server interacting with multiple instances of these two clients. Since the application would be tested locally, the provision of an IP address input has been omitted from the design.

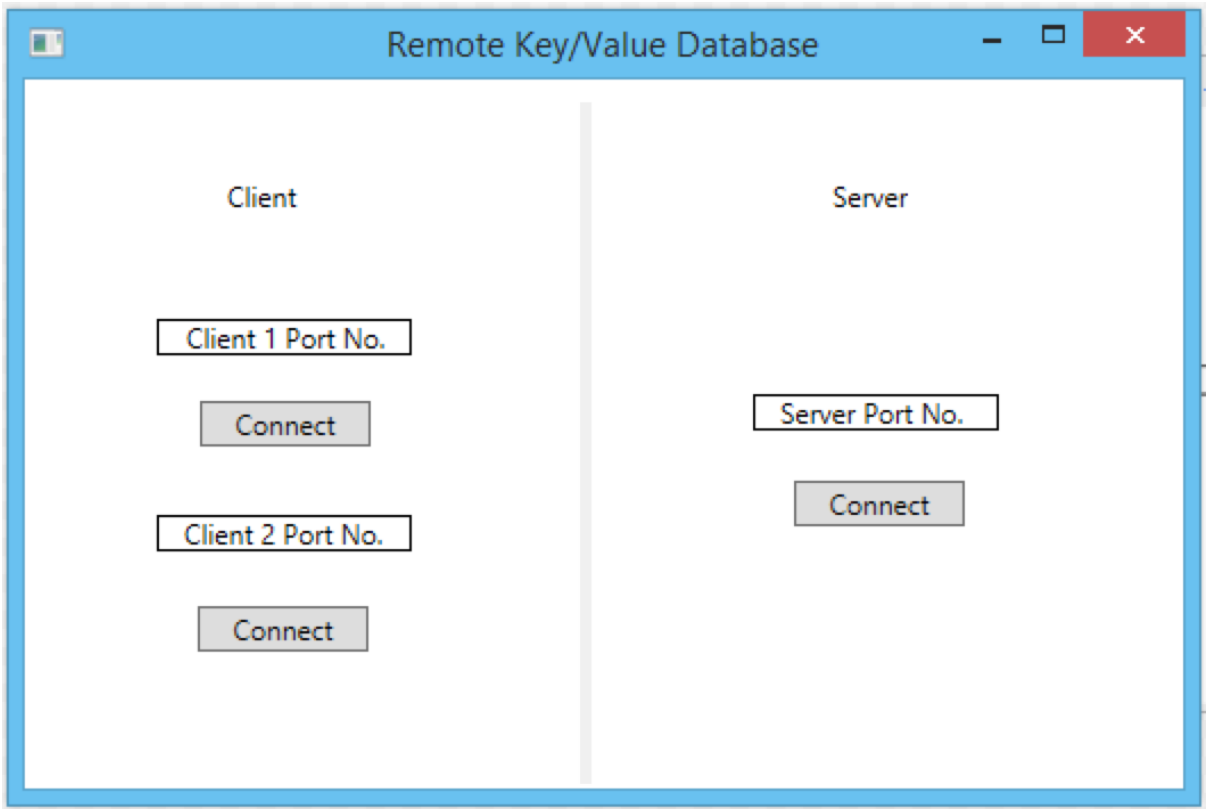

Figure 3. Connection Window for Remote Key/Value Database

3.2 Client Operations Window

The Client Operation window provides tabs for database operations including addition/deletion/edit of records, query processing and persistence of database into XML file and vice-versa.

DB Operations Window provides the user with a window suitable for addition, deletion and edit of records. In case of Addition and edit, the user is required to enter the Key to add/edit, the metadata information that consists of name, description and children of the Key and lastly the payload or the value of the Key. In case of deletion, the user is required to enter the key that he/she wants deleted.

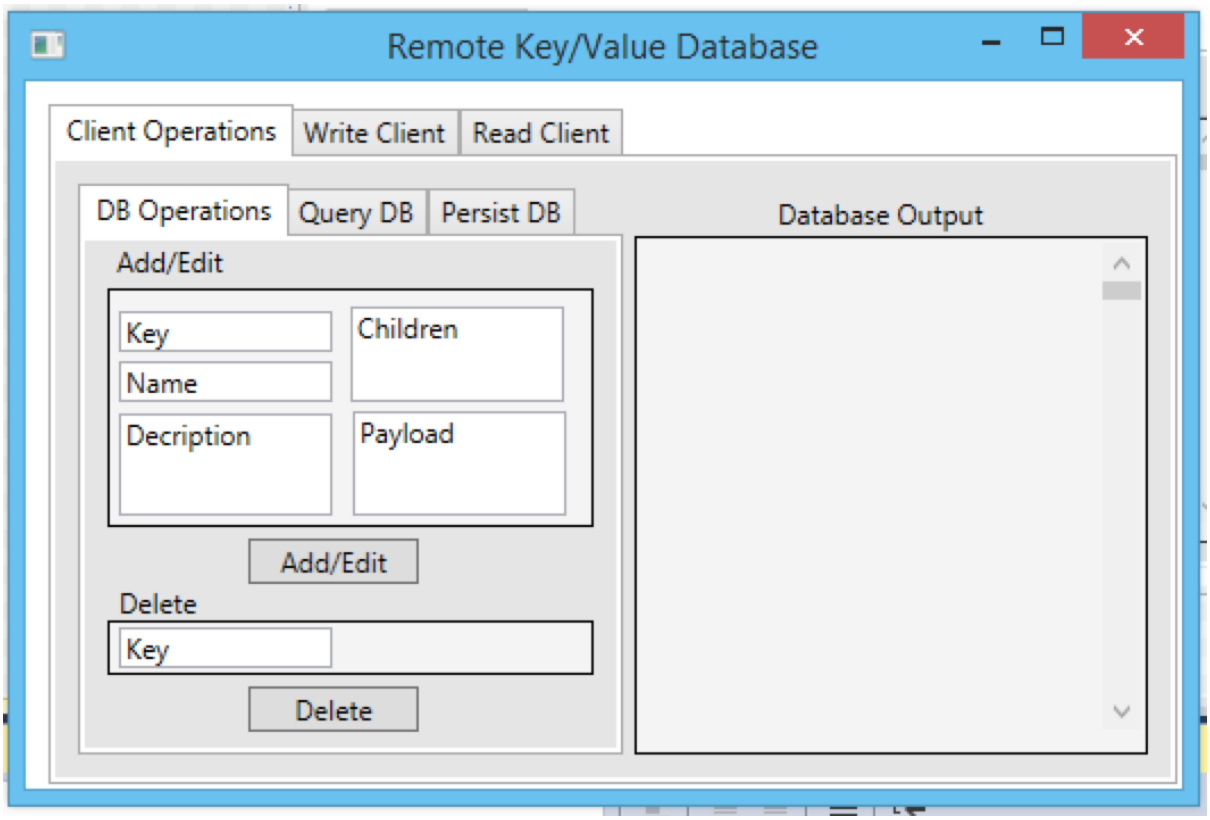

Figure 4. DB Operations Window for Remote Key/Value Database

QueryDB Window provides a window for query processing. It consists of various radio buttons that represent a pre-set query defined on any Key. When the user selects a radio button, the query matching the radio button is fired and the result is displayed on the 'Database Output' screen.

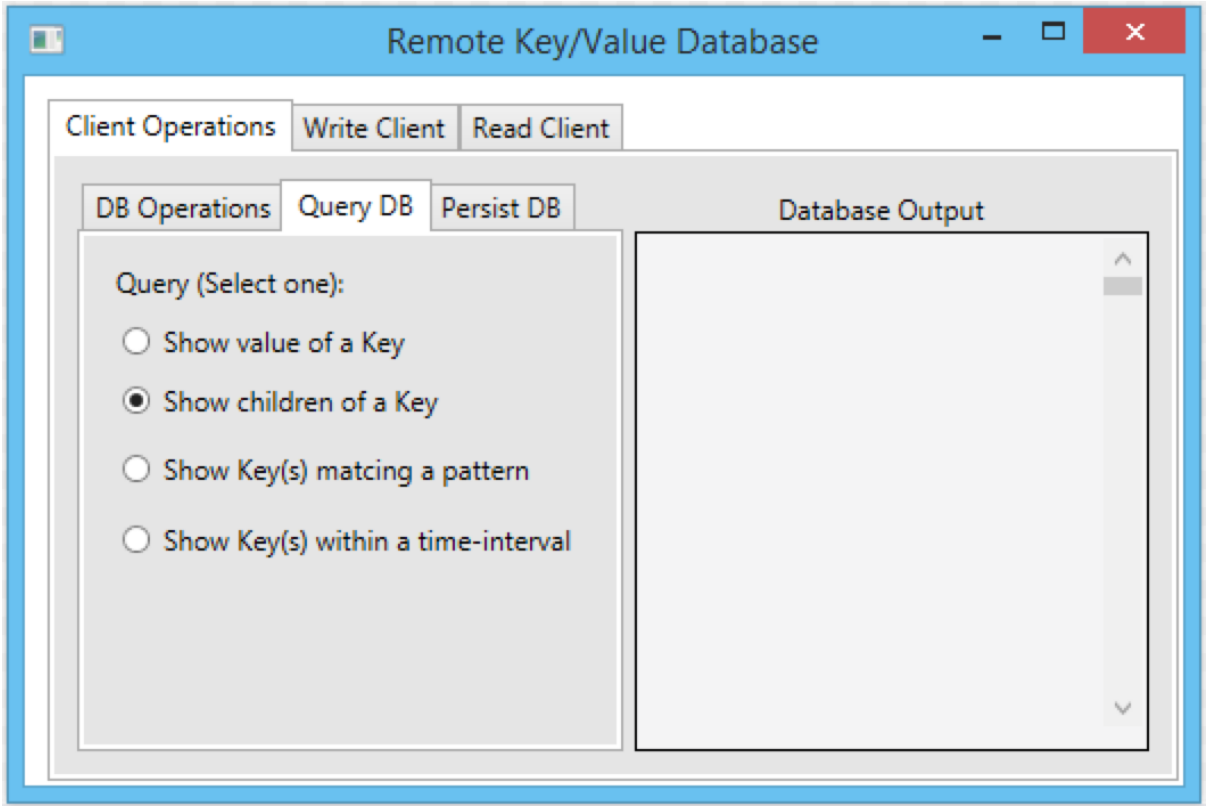

Figure 5. Query DB Window for Remote Key/Value Database

PersistDB Window provides a window to support the persistence of database contents from in-memory database to an XML file. Unpersist causes the data in the XML, in the form of various XML elements to be restored in the form of a Key/Value database. The UI view provides a button to the said functions.

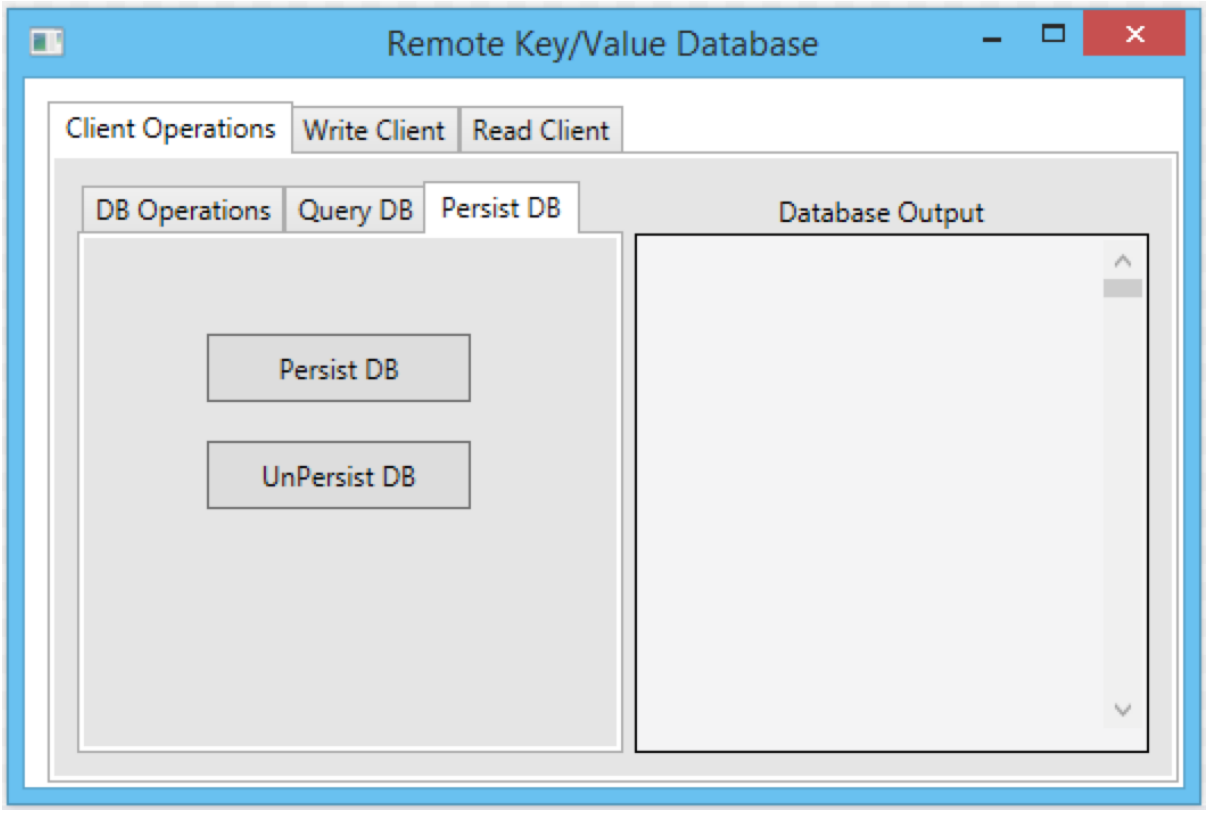

Figure 6. Persist DB Window for Remote Key/Value Database

3.3 Write Client Window

The Write Client window provides a sample UI view to the Write Client. Its functions include sending a pre-fixed number of messages to the remote server and displaying the contents written to the database in the database output screen.

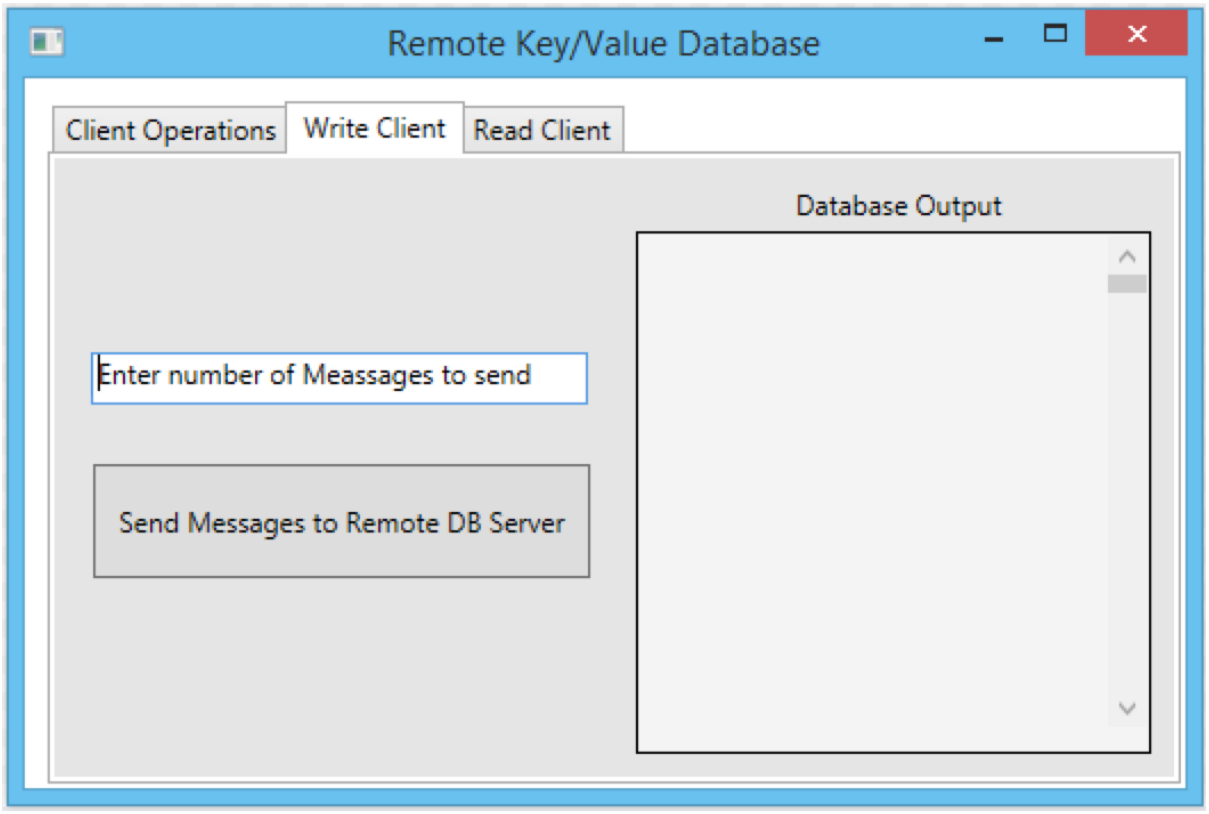

Figure 7. Write Client Window for Remote Key/Value Database

3.4 Read Client Window

The Read Client window provides a sample UI view to the Read Client. Its functions include display of the response query messages sent to the server.

It also includes a Check box that gives the users an option to view the responses in a detailed manner or a concise manner.

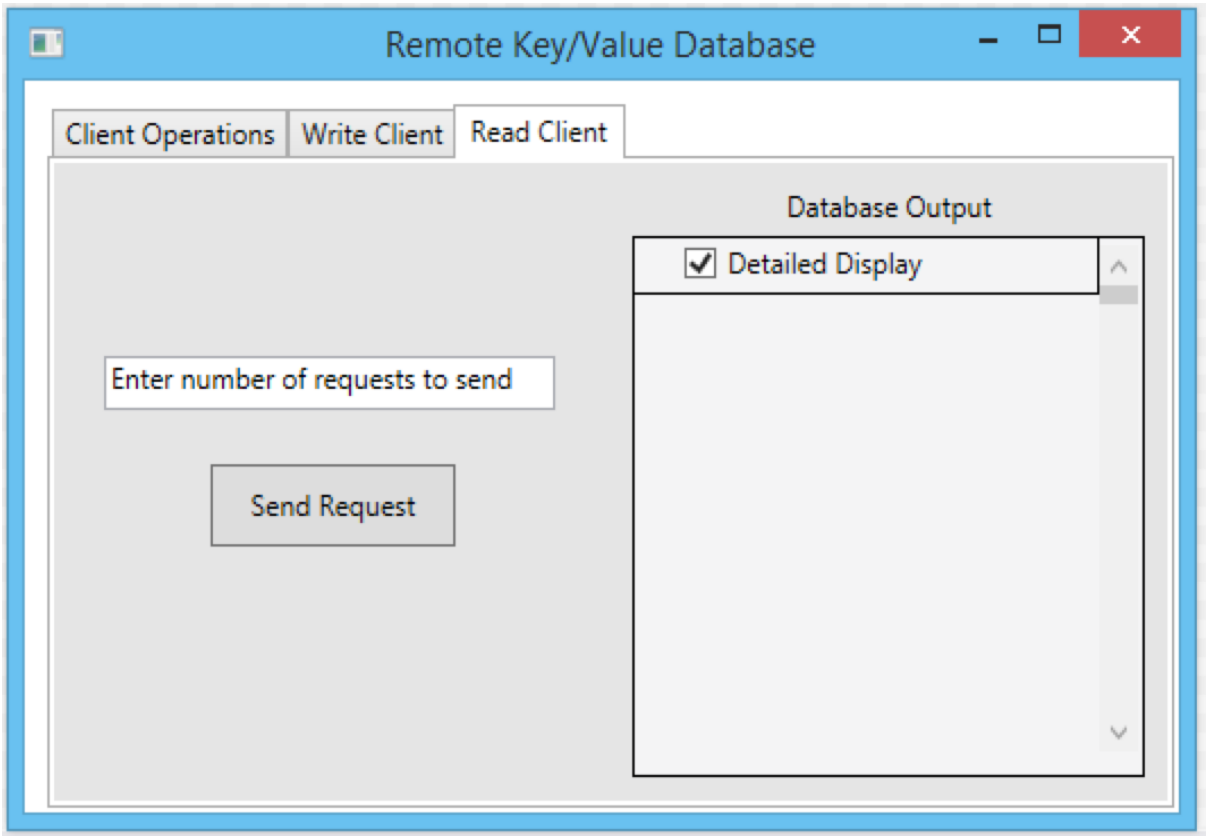

Figure 8. Read Client Window for Remote Key/Value Database

4 Architecture

The structure of the Remote Key/Value database consists of various components that perform complex operations with the help of eachother or independently. The following diagrams depicts the various components of the Remote Key/Value database and the flow of data.

4.1 Client Partitions

The Client side includes Client GUI, Message Parser and Communication Modules packages to support the remote Key/Value database operations. The Performance module mentioned in the architecture may be eventually described in one of the Client Package structures.

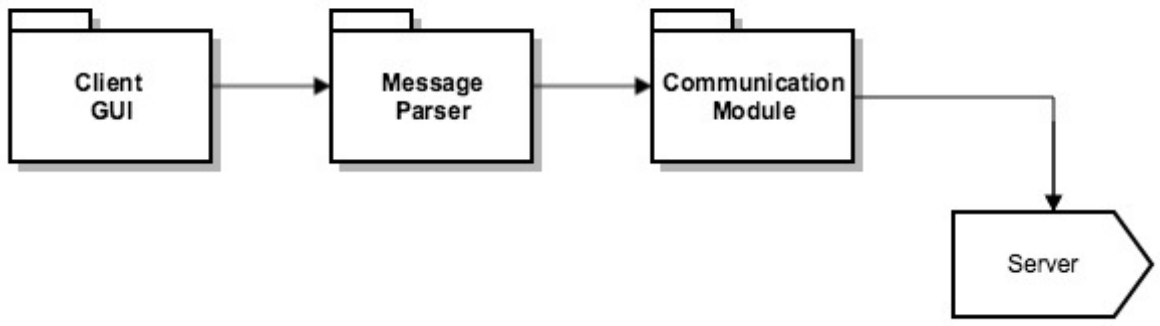

Figure 9. Client Package Diagram for Remote Key/Value Database

4.1.1 Client GUI Package

The Client GUI package contains the classes to define the UI of the application. It consists of a Windows Presentation Foundation (WPF) project that would outline the various windows that the user would view when he/she uses the application.

Responsibilities: The main responsibility of the Client GUI is to provide fucntions to carry-out various operations on the screen and delegate the actual database logic to a separate class.

Interaction with other packages: The client GUI interacts with **Message Parser** and the **Communication Module.**

4.1.2 Message Parser Package

The Massage Parser provides a Class that provides a definite structure to the messages going out of the Client through the Blocking queue. The structure of the Message might look like this,

```
[DataContract]
 public class Message
 {
  [DataMember]
  public string fromUrl { get; set; }
  [DataMember]
  public string toUrl { get; set; }
  [DataMember]
  public string content { get; set; }
 }
```

The Message structure involves a fromURL and toURL that would help to track the owner and the recipient of the package. The string content field may be used to populate an XML message format.

Responsibilities: It sole responsibility is to parse the messages from the client using suitbale Windows Communcation Foundation (WCF) Data Contract.

Interaction with other packages: It interacts with **Client GUI** to gain the messages and the **Communication Module** to send the messages through the Blocking queue.

4.1.3 Communication Module Package

The Communication Module, as the name suggests, helps in the communication of messages between the Client and the Database Server. The Communication Module would hold Interface for communication that would hold a Data Contract. This could be implemented in C# as,

```
[ServiceContract]
 public interface ICommService
 {
  [OperationContract(IsOneWay=true)]
  void PostMessage(Message msg);
  Message GetMessage();
 }
```

```
[DataContract]
 public class Message
 {
  [DataMember]
  Command cmd = Command.DoThis;
  [DataMember]
  string body = "default message text";

  public enum Command
  {
   [EnumMember]
   DoThis,
   [EnumMember]
   DoThat,
   [EnumMember]
   DoAnother
  }

  [DataMember]
  public Command command
  {
   get { return cmd; }
   set { cmd = value; }
  }

  [DataMember]
  public string text
  {
   get { return body; }
   set { body = value; }
  }
 }
```

Responsibilities: It defines a sender and a receiver blocking queue. A **blocking queue** is a queue that block when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other

thread inserts an item into the queue. This ensures the messages are sent in a thread-safe manner and no message is lost.

Interaction with other packages: The communication module interacts with the **Communication Module** at the Server end.

4.2 Server Partitions

The Server side includes Communication Modules, Message Parser, Server Engine, Query Engine, DB Extensions, DB Engine, DB Element, DB Factory and XML Parser packages to support the remote Key/Value database operations. The Performance module mentioned in the architecture may be eventually described in one of the Structure Package structures.

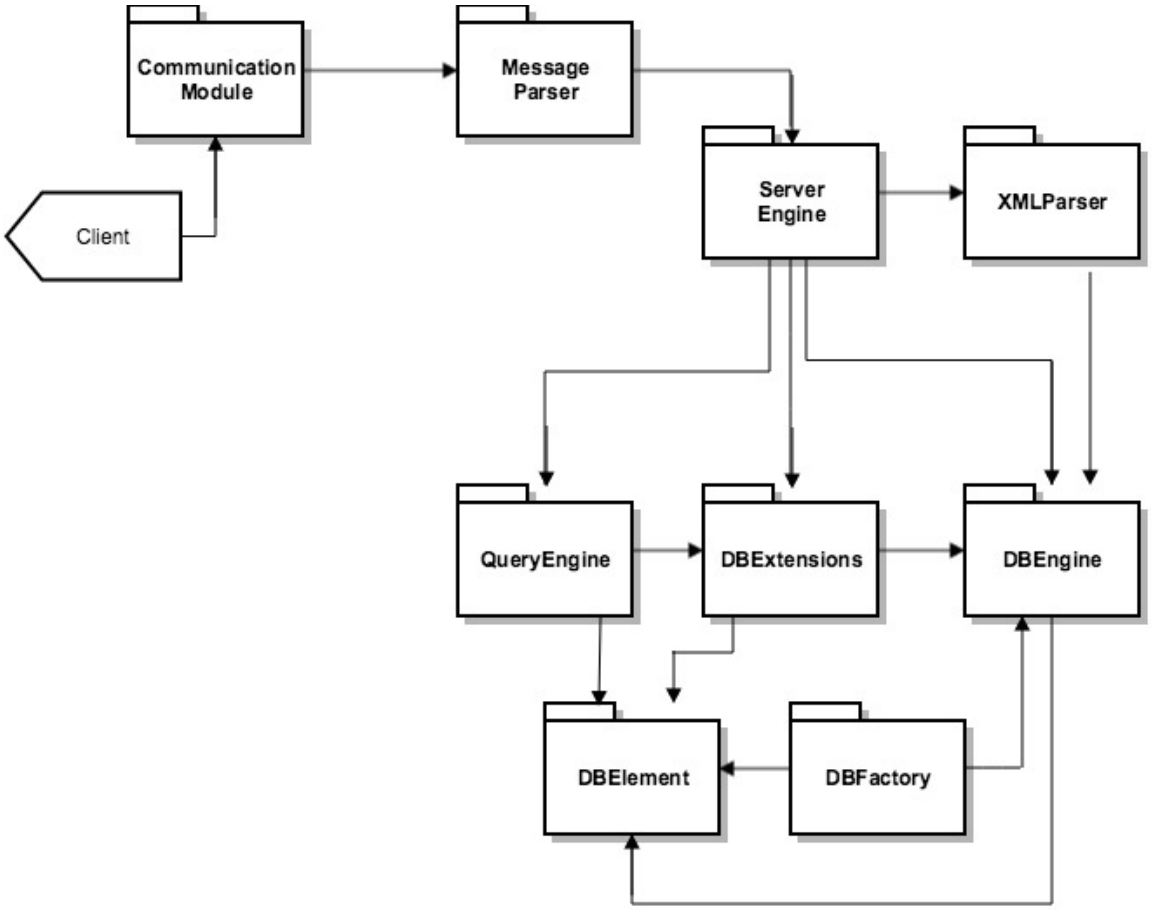

Figure 10. Server Package Diagram for Remote Key/Value Database

4.2.1 Communication Module

The Communication Module, as the name suggests, helps in the communication of messages between the Client and the Database Server.

Responsibilities: It defines a sender and a receiver blocking queue. A **blocking queue** is a queue that block when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue. This ensures the messages are sent in a thread-safe manner and no message is lost.

Interaction with other packages: The communication module interacts with the **Communication Module** at the Server end.

4.2.2 Message Parser

The Massage Parser provides a Class that provides a definite structure to the messages going out of the Client through the Blocking queue.

Responsibilities: It sole responsibility is to parse the messages from the client using suitbale Windows Communcation Foundation (WCF) Data Contract.

Interaction with other packages: It interacts with **Client GUI** to gain the messages and the **Communication Module** to send the messages through the Blocking queue.

4.2.3 Server Engine

The Server Engine package acts as the Executive package, that delegates to the various packages with respect to the application logic.

Responsibilities: The responsibilities of the Server Engine package consists collecting input messages from the clientand delegating to the appropriate package.

Interaction with other packages: It needs to interact with all the other packages of the server system that are vital to the requirement specification. This includes packages like, **DBElement, QueryEngine, DBEngine, XML Parser DBFactory and DBExtensions.**

4.2.4 DB Element Package

The DBElement package is used to define items or instances, which shall consist of metadata and an instance of a generic type. This essentially forms the key/value pair. Each database Value has structured meta-data and an Instance of the generic type. This implemented using C#, would look like this,

```
public class DBElement<Key, Data>
 {
  public string name { get; set; }          // metadata
  public string descr { get; set; }         // metadata
  public DateTime timeStamp { get; set; }   // metadata
  public List<Key> children { get; set; }   // metadata
  public Data payload { get; set; }         // data
 }
```

Responsibilities: The sole reponsibility of the DBElement package is to provide package to define the structure of the instances or items, which in this case consists of a key and a value that shall contain certain metadata fields such as, A name string, A text description of the item, A DateTime string recording the date and time the value was written to the database, A finite number (possibly zero) of child relationships with other values.

Interaction with other packages: The DBElement package would interact with **DBEngine, DBFactory, QueryEngine** and **DBExtensions** package.

4.2.5 Query Engine Package

The Query Engine package prescribes over the vital query processing tasks of the system. This module includes the design of the queries, which shall be used to fulfill the requirement specifications. The Query Engine may utilize Predicate Lambda functions to support query processing. This implemented using C#, would look like this,

Query to display children of a key:

```
Func<string, bool> qp = (string key) =>
    {
     DBElement<string, PL_ListOfStrings> qelem;
     if (db.getValue(key, out qelem))
```

```
    if (qelem.children.Count() > 0)
      return true;
   return false;
  };
```

Responsibilities: It satisfies the requirement to support queries for the value of a specified key, the children of a specified key, the set of all keys matching a specified pattern which defaults to all keys, all keys that contain a specified string in their metadata section and all keys that contain values written within a specified time-date interval.

Interaction with other packages: It interacts with **ServerEngine** package to provide the input, **DBEngine** package in order to support query processing using the Dictionary and **DBFactory** in order to provide a virtual database for faster query processing.

4.2.6 DB Engine Package

The database engine package defines the C# Dictionary class member. C# class representing the Database Engine class might look like this,

```
public class DBEngine<Key, Value>
 {
  private Dictionary<Key, Value> dbStore;
 }
```

Responsibilities: The sole responsibility of a Database Engine package is to provide a class to hold the Dictionary<Key,Value> member. It is also used to populate its noSQL database instance through an API provided by the DBEngine package.

Interaction with other packages: The Database Engine package interacts with the **ServerEngine** package to ackowledge the input, the **QueryEngine** package to process queries using the db connection, the **XML Parser** since it needs to store data contents from the database into the XML file, the **DBFactory** since the DBFactory package is essentially a virtual database.

4.2.7 DB Factory Package

The Database Factory package supports the creation of a new immutable database constructed from the result of any query that returns a collection of keys. It thus provides a virtual database that can be used to query upon by means of formation of compound queries, thus eliminating the need to query the actual database each time a key is required to be queried, which in-turn will save time and thus lead to faster query processing which is crucial in an environment dealing with large data sets.

Responsibilities: The Database Factory package is responsible for providing this virtual database as mentioned above.

Interaction with other packages: The Database Factory package interacts with **DBEngine** package to support Dictionary look-up and **QueryEngine** to support the link to all the queries mentioned under the requirement specification.

4.2.8 XML Parser Package

The XML parser package provides a couple of Classes to provide Database persist, unpersist and scheduled persist functionality. It also helps in data sharding.

Sharding, divides the data set and distributes the data into multiple files or shards. This in-turn reduces the time taken to process a query, since the search is limited to a shard instead of the entire database.

Shard Key, is the criteria based on which shards are created or partitions are created. Shard key is either an indexed field or an indexed compound field that exists in every document in the collection (or database collection). This might be accomplished using either **range based partitioning** or **hash based partitioning**.

Responsibilities: The main responsibility of the XML Parser package is to provide a class that deals with storing of database content from in-memory database to an XML file. **XElement** and **XDocument** are used to convert the DB content into an XML format. It also does unpersistence, which involves converting data back from an XML format into the database as Key/Value elements.

The XML Parser package also provides for a Persist Scheduler class, this class is used to automatically persist the database contents into an XML file based on an auto-timer.

Interaction with other packages: It interacts with the **Server Engine** and **DBEngine** package.

4.2.9 DB Extensions Package

The DB Extensions package is another crucial package that provides extension methods to display or print the DB Element on the console.

Responsibilities: Its reponsbility includes prinitng or displaying the results of the queries, the key/value pairs or other important information required to accurately depict the requirement fulfillment.

Interaction with other packages: It interacts with the **DBEngine, DBElement, DBFactory, QueryEngine** and **ServerEngine** package in order to support displaying the Database Elements on the console.

5 Application Activities

5.1 Activity Diagram for Client

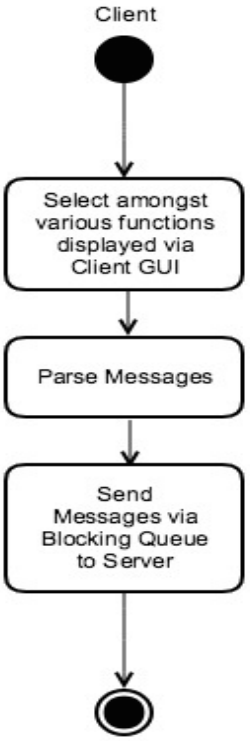

Figure 11. Client Activity Diagram for Remote Key/Value Database

5.2 Activity Description

Selection of a function via Client GUI

The activity defines the selection of a function in the Client GUI. This may include activites mentioned previously in the sample UI, that is, DB Operations, Write Client operations and Read Client operations.

Parse Messages

The process flow would then be directed onto the Message Parser package, which as the name suggests would parse the different type of messages and consolidate them into a structured format for message passing over the communication channel.

Send Messages to Server side Communication channel (Blocking queue)

Once the messages have been formed they need to be tranmitted using a suitable WCF Data Contract to the server. This is done using the sender

blocking queue. The blocking queue ensures delivery of the messages to the server side receiving blocking queue.

5.3 Activity Diagram for Server

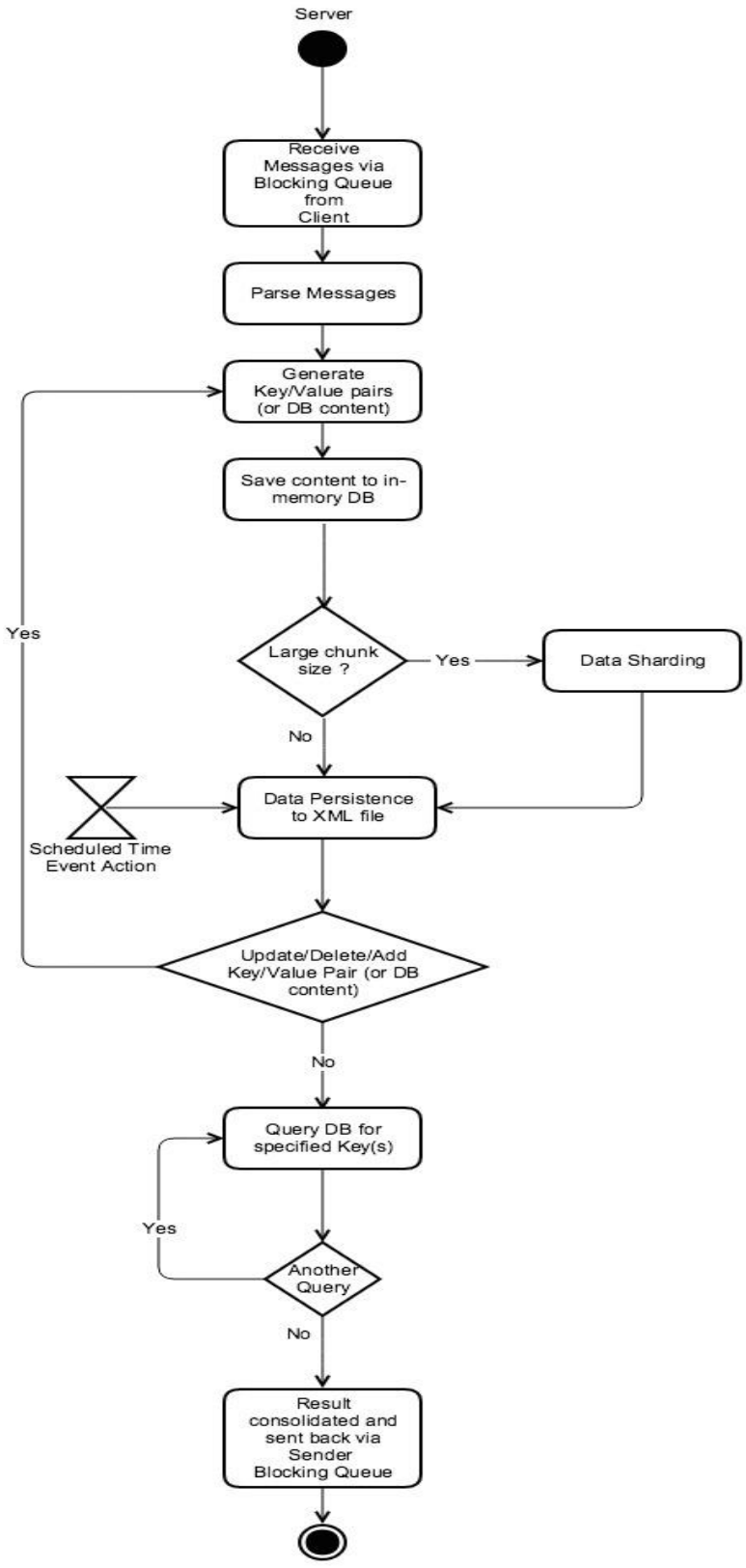

Figure 12. Server Activity Diagram for Remote Key/Value Database

5.4 Activity Description

Receive messages via the Client side Communication Channel (Blocking queue)

This activity defines the receipt of messages sent by the client using a WCF Data Contract over the blocking queue. The messages on the server-side are received by the receiving blocking queue defined in the communcation module of the server.

Parse Messages

Once the messages have been received, they are parsed on the server side in order to transfom them into elements that may be stored and re-used by the database.

Generate Key/Value pairs (or DB content)

This activity or task supports creation of items or instances described by metadata and holding some generic type of value.

Save content to in-memory DB

The process to save the contents or in this case the instances into the database. In case the data set is too large, it is sent out to Shard. In other cases where the data is not large it is sent out to be persisted to an XML file.

Data Sharding

In case of large data chunks determined by specific size considerations, the large data is sharded or divided into smaller pieces and then persisted onto XML files.

Data Persistence to XML file

Data is stored in XML file. This is followed by a decision node which supports the addition, deletion or updation of any instance record in the database. If either is wished, the process is repeated again starting form generation of key/value pairs, else the data flow is allowed to move further down the activity diagram to proceed with the remaining tasks.

Scheduled Time Event Action

This is an event that occurs in a fixed schedule, in this case the data persistence into the XML files is a process required to occur after every specfic time interval.

Query DB for specified key(s)

This activity supports simple and compound query processing. If further query result is wished then the process is repeated, else the results are printed out to the console.

5.5 Activity Diagram for Read Client

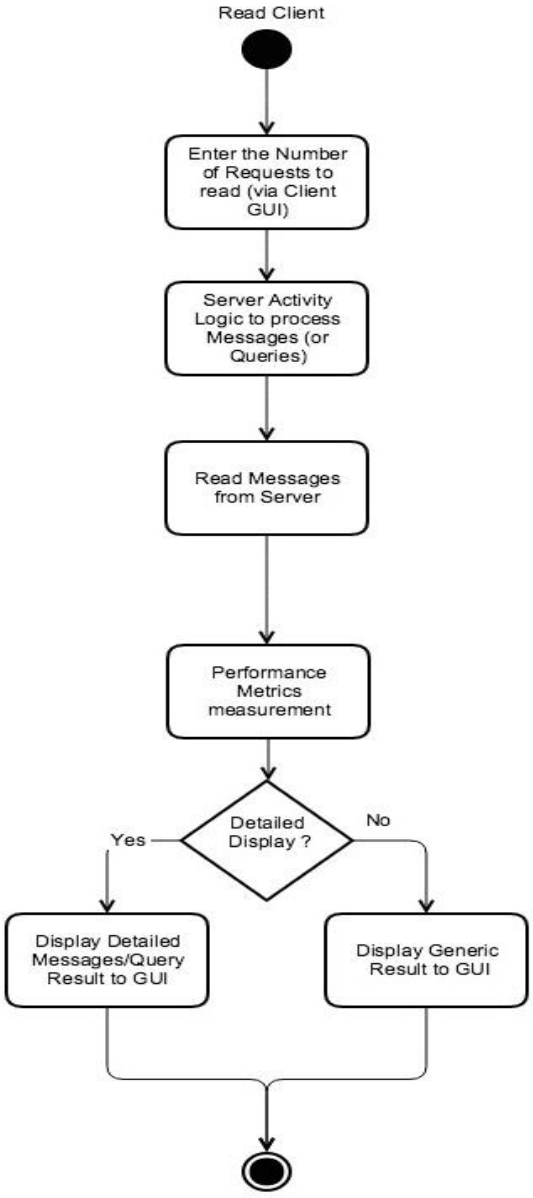

Figure 13. Activity Diagram for Read Client

5.6 Activity Description

Enter Requests to read (via Client GUI)

Initially, the read client prompt the user to enter the number of messages the user wants it to process and display.

Server Activity logic for request processing

The database server processes the messages that are requested from the database and sends it through the communication channel back to the Client.

Read Messages from the Server

Messages sent from the database server are parsed into a format that can be displayed onto the Client display GUI.

Performance metrics measurement

The time taken for the database to process each request as a measure of unit time is calculated with the help of the performance metric module. It may also perform calculations for the time taken to perform the entire operation. The result are then stored and displayed via console or GUI.

Display

If the user asks for a detailed report of all the messages or queries read from the database, he/she is directed to a more informative view of the messages. Else, a generic view is displayed on the Client GUI.

5.7 Activity Diagram for Write Client

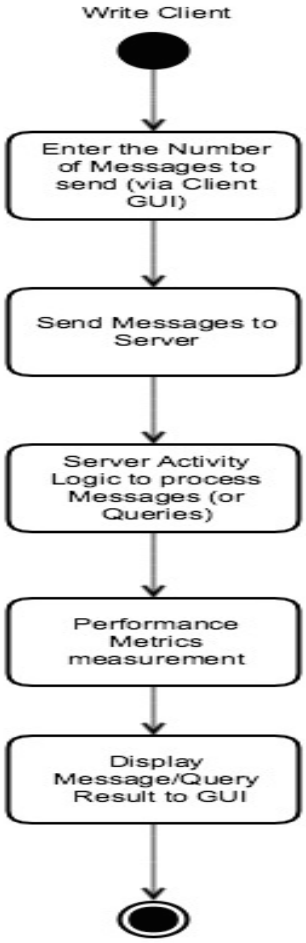

Figure 14. Activity Diagram for Write Client

5.8 Activity Description

Enter Number of Messages to send (via Client GUI)

Initially, the write client prompt the user to enter the number of messages the user wants it to process and display.

Send Messages to Server

The messages are loaded from an XML file in random fashion, i.e. this may include various different types of messages randomly selected from a base of messages defined in an XML file. These messages are parsed and sent to the database server.

Server Activity Logic

The database server performs its logic to store the message contents into the database. In case of a query it would process each query based on the type of query. Once the processing is completed a result of the database output is sent back to the Client via the communication channel.

Performance metrics measurement

The time taken for the database to process each request as a measure of unit time is calculated with the help of the performance metric module. It may also perform calculations for the time taken to perform the entire operation. The result are then stored and displayed via console or GUI.

Display

An output of all the messages or queries performed on the database is displayed on the Client GUI.

5.9 Activity Diagram for Sharding

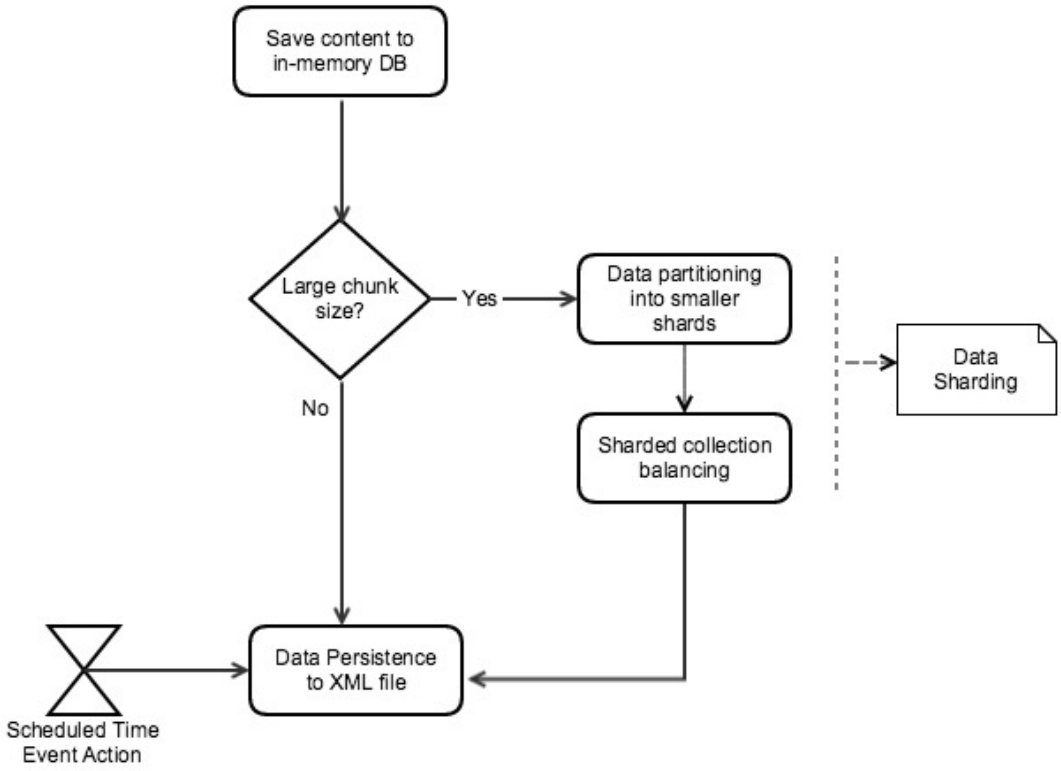

Figure 15. Activity Diagram for Sharding in Key/Value Database

5.10 Activity Description

Save content to in-memory DB

The process to save the contents or in this case the instances into the database. In case the data set is too large, it is sent out to Shard. In other cases where the data is not large it is sent out to be persisted to an XML file.

Data partioning into smaller shards

In case of large data chunks determined by specific size considerations, the large data is sharded or divided into smaller pieces and then persisted onto XML files.

Sharded collection balancing

The collection of sharded data pieces are balanced to make sure any shard does not hold unnecessary large amount of data, thus defeating the purpose of Sharding all together.

Data Persistence to XML file

Data is stored in XML file.

Scheduled Time Event Action

This is an event that occurs in a fixed schedule, in this case the data persistence into the XML files is a process required to occur after every specfic time interval.

5.11 Activity Diagram for Querying

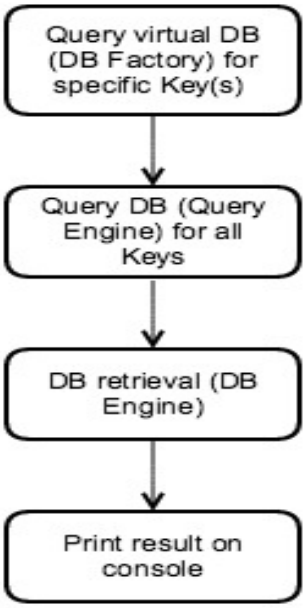

Figure 16. Activity Diagram for Querying in Key/Value Database

5.12 Activity Description

Query virtual DB (DBFactory) for specific key(s)

This activity or task supports the requirement of creation of a new immutable database constructed from the result of any query that returns a collection of keys

Query DB (QueryEngine) for all keys

This activity includes querying the actual database to support the requirement specification.

DB retrieval (DBEngine)

This activity supports the database storage using the C# Dictionary.

Print result on the console

The activity prints or displays the query results on the console window.

6 Critical Issues

6.1 Database (or Query) Performance

Issue: An important part of this project is to consider the query performance of the Clients. Query performance is the time metric measured once a successful query is implemented. This performance would differ with different types of messages. Though the type of payload we consider in the following project may be fixed to List of Strings but it may vary in terms of its construction. For example a message payload might be a list of single worded strings, or a list of paragraphs of strings. This causes an issue with respect to the query performance of each message.

Solution: A possible solution for faster query retrieval is to ensure faster serialization and de-serialization. Serialization is a process of converting an object into a stream of bytes to send over a network or store in a database.

Design Impact: Serialization entails accessing the class' objects whose fields need to be serialized. Initially it is inspected through reflection in order to generate fields that serialize the declared fields. This process is made faster if objects can be accessed directly. Ensuring the class to be in the same package as the serialized object's class could do this.

6.2 Concurrent Access to Remote Key/Value Database by Read and Write Clients

Issue: The application involves multiple Read and Write Clients to interact with the database simultaneously, while producing results in a fairly small amount of processing time. Owing to this nature of the application, there might be a contest as to who would access the database first, considering two or more Clients run on parallel threads at the same time.

Solution: A trivial solution of this issue is the implementation of blocking queues at the Client and the Server end.

Design Impact: The solution would entail building a sender and receiver blocking queue at the Client as well as the Server end. The blocking queue would act as a queued buffer that would hold and release the messages in a linear fashion to the other end, i.e. the Client or the Server.

6.3 Message passing using Blocking queue

Issue: Although implementation of blocking queue solves the issue of concurrent access, there may be situations when the there are an overwhelming number of messages in the blocking queue, more than what the Server has been designed to handle. This leads to an issue with message passing using the Blocking queue.

Solution: A dynamic or open-ended blocking queue might be helpful in a situation of heavy load of messages, but eventually it might be a good idea just to let the system run its course and re-run the queries when the load is less.

Design Impact: A dynamic blocking queue may be designed using C# and might look like this,

```
public class BlockingQueue<T>
  {
    private Queue blockingQ;
    object locker_ = new object();
    //----< enqueue a string >-------------
    public void enQ(T msg)
    {
      lock (locker_)  // uses Monitor
      {
        blockingQ.Enqueue(msg);
        Monitor.Pulse(locker_);
      }
    }
 }
```

6.4 Concurrent Write to the same Key

Issue: There might be a situation where two Clients try to write to the same Key. That is, when the Key-Value pair is initially being populated, two separate clients might have the same Key but assign different value to the Key. This causes a critical issue since it could lead to inconsistency within the database.

Solution: The situation can be handled in two ways, either prevent this issue to occur and thus modify the design accordingly or, the value may be overwritten each time a new Client tries to write to an existing Key.

Design Impact: In case of prevention, implementation would consist of not allowing a client to add value to a key that is already in the database. In the other case, the existing value and metadata of the key would just be overwritten to the value set by the latest Client.

6.5 Embed application logic in Client GUI

Issue: As mentioned in section 4, the UI of the application contains a considerable amount of control, and would be the eventual executive of the application; this could lead to the temptation of dumping all the code behind each dialog window control to the respective button handler. This is a malpractice and even though it might sound easier to do, it usually causes the system to slow down considerably.

Solution: The Client GUI in the application would act as nothing but a façade to the further complex sub systems that it hides. This leads to a light UI package structure with less amount of code, thus making it faster and adhering to code standards.

Design Impact: The Client GUI package would act as a delegate, calling other respective functions based on which control button is accessed by the user. Thus the Client package structure would not contain the complex sub system design logic that actually carries out the bulk of the operations in the application.

6.6 Client crashes after sending a request

Issue: There can be a scenario where a client can crash after it has sent the request but not received response.

Solution: Even though client might have crashed but server would have started the database operation, or worst finished it but failed to send the response. This will lead to wastage of server bandwidth. Server can implement a failed queue and store the result until client comes up. Once the client is ready it can send out the results to it.

Design Impact: As the solution suggested, the design impact would accost the implementation of a queue that would store the result until the Client wakes.

This could be implemented as a separate queue implemented in the Communication Module package structure of the Server.

6.7 Key-Value deletions

Issue: With the deletion of Keys, comes the question of deleting its subsequent children as well or not. Deletion of the children of a particular key that is deleted might cause various application inconsistencies and hence is an issue of concern.

Solution: Eventual deletion of a child key due to the deletion of its parent might cause a query to point to a key that does not exist in the database. In order to prevent the application from crashing at this point of time, exception handling must be carried out.

Design Impact: Provision of a simple try-catch block at the point where the code accesses the database should be appropriate to ensure the application does not crash. It might return a false value at this instant, but the application is prevented from total failure.

6.8 Dictionary look-up

Issue: Dictionary is a constant time operation if the key of the key/value pair being searched is provided. However, it turns into a linear search through all the key/value pairs if a specific key is not provided. Thus leading to large time taken for query processing.

Solution: A solution to avoid such an escalated time of search is to divide the data set into smaller pieces or files called Shards, this process is called Sharding.

Design Impact: Provision of a Class that will be utlized to break the large data set into smaller pieces for faster query processing.

6.9 Version Control

Issue: Version control is used to manage medium to large software systems in order to provide them with a sense or organization and consistency. In this case, multiple users may try to simultaneously build of check-in changes in the

code they have developed, thus it may cause inconsistent database data if not handled carefully.

Solution: The solution entails ensuring that the data committed to the final version in the server is consistent and accurate.

Design Impact: This can be achieved by th creation of a module that ensures a versioning of the build each time the code is checked out to make changes by any user.

7 Prospective Application

After the completion of the project driven by this operation concept document, we shall work on a concept and implement remote access to the noSQL prototype via message-passing communication.

Finally we develop architecture, for a data management service in a large Software Development Collaboration Environment using the noSQL model we created in the earlier projects. The architecture of this future project is mentioned below.

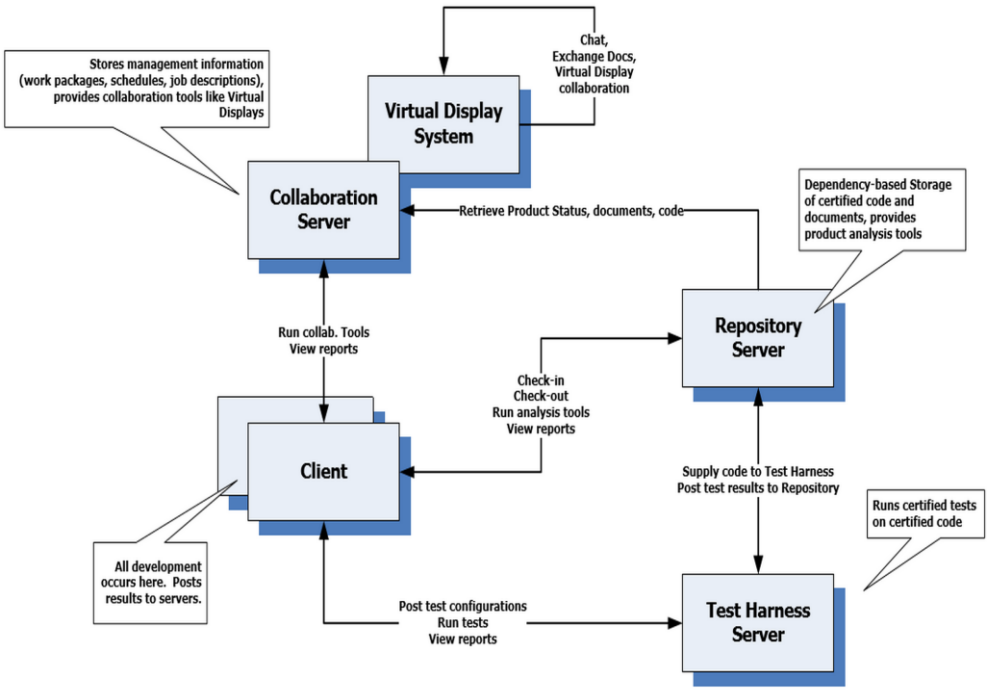

Figure 17. Diagram depicting Software Collaboration Federation(SCF)

8 Conclusion

The Remote Key/Value Database provides a useful application as a noSQL database implemented in a distributed environment, whilst retaining the advantages of large data storage, quick query processing, instant data availability, scalability and agility. The remote key/value database finds its utility in users like developers, instructors, software systems, organizations and industries involveing supply chain management, health care analytics and social media. Robust and functional structure of its various components lead to its utility in other functional languages as well. This Operational Concept Document precisely defines sample UI screens, the task or activity description involved in the project and also lists a number of possibe critical issues that may be experienced while constructing the application. The solution and the design impacts listed with the critical solutions must be adhered to in order to ensure an able and successful application.

9 References

1. Fawcett, Jim. *CommPrototype*. .Net Framework ed. C# Project. 2015.
2. Fawcett, Jim. "Code Artistry - No SQL Databases." Blog NoSql. Accessed September 6, 2015. http://ecs.syr.edu/faculty/fawcett/handouts/webpages/blogNoSql.htm.
3. Fawcett, Jim. "SG - OCD." SG - OCD. Accessed September 10, 2015. http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm.
4. “MongoDB”. Accessed September 10, 2015. http://docs.mongodb.org/master/MongoDB-sharding-guide.pdf.
5. "NoSQL Databases Defined & Explained." Planet Cassandra. January 10, 2014. Accessed September 8, 2015. http://www.planetcassandra.org/what-is-nosql/#nosql-explained.
6. "NOSQL Database Explained." NoSQL Databases Explained. Accessed September 10, 2015. https://www.mongodb.com/nosql-explained.