Mobile Data as a Service

Project Report (v2.0)

By
Sabyasachi Sengupta
Tarini Pattanaik
Tuyen Ly

Project Academic Advisor
Dr. Jerry Gao

Project Industry Advisor
Dr. Wei-Tek Tsai

March 13, 2013

APPROVED FOR COLLEGE OF ENGINEERNG

Dr. Jerry Gao, Academic Advisor, Department of Computer Engineering, SJSU

Dr. Wei-Tek Tsai, Industry Advisor, Arizona State University

Dr. Lee Chang, Director, Department of Computer Engineering, SJSU

ABSTRACT

Mobile computing is one of the emerging trends that will be pervasive to the technology world in the coming decades. Not only mobile devices have the potential to become the primary computing device for users in future, they, coupled with recent advancements in the Internet and cloud technologies, can provide users with fast and easier means to access information. Modern day mobile phones and tablets from leading vendors such as Apple, Google, Motorola and others can potentially replace laptops and heavy weight counterparts like desktop computers. Data storage in general, and databases in particular present some interesting challenges that need to be addressed so that these devices can be used in that fashion. Owing to its inherent mobile nature, a variety of database services namely, seamless connectivity; data synchronization, query management, disaster recovery, security etc. differ significantly between a traditional system and a mobile device. The surge of cloud computing over the past few years have opened new avenues for leverage of its distributed compute resources, automatic scaling and storage services for meeting the ever-growing needs of mobile computing. In the current work we propose a new paradigm called *Mobile Data as a Service (MDaaS)*. The notion of *Data as a Service* is not new. It refers to the general capabilities of providing resiliency, multi-tenancy and other database service oriented architecture for accessing data in the traditional desktop world. As mobile clients require to constantly synchronizing their copies of databases with the master data, the choice of cloud seems obvious. MDaaS is a new cloud solution that can address the above-mentioned needs of databases residing in mobile compute devices that a typical cloud oriented database does not provide. It hosts special database acceleration software that will be used by both incoming and outgoing traffic in the cloud for ensuring a scalable, and highly available database solution even when mobile clients are on the move. We believe this paradigm is not only necessary but is essential in the mobile database design in the coming decade.

Table of Contents

1 Introduction

1.1 Project Goals and Objectives

The purpose of this project is to design and develop a new cloud based solution called *Mobile Data as a Service (acronym MDaaS)* that would be responsible for improving transaction failover and resiliency, optimizing bandwidth and providing a number of resiliency features for mobile computing devices. This document is a detailed report, which examines the possible use case scenarios that lead to the engineering requirements, and enumerates a high level design of this product. It also discusses various considerations about implementation and test plan that needs to be executed when this product is taken up for implementation.

1.2 Background, Problems and Motivation

A *mobile database* is defined as "*a database that resides on a mobile device such as a PDA, a smart phone, or a laptop. Such devices are often limited in resources such as memory, computing power, and battery power*" [11]. With the exponential growth of mobile users in the past decade, such databases are being used in mobile devices in order to temporarily store data and serve the applications when required. However, such databases often need to communicate with the outside world, as they need to synchronize their data. As the very nature of mobile devices is that they can get frequently disconnected or have low bandwidth and processing power, resiliency of transactions cannot be guaranteed. As a result, the applications need to frequently restart database transactions. This leads to performance overhead.

In addition, the distance of the master database from the mobile devices often translates to the latency and throughput. Today many leading database vendors are hosting their data in the cloud. A *cloud database* management system (CDBMS) is defined to be a "*distributed database that can deliver a query service across multiple distributed database nodes located in multiple data centers, including cloud data centers. Querying distributed data sources is precisely the problem that businesses will encounter as cloud computing grows in popularity. Such a database also needs to deliver high availability and cater for disaster recovery*." [23]. However, the cloud databases are specifically tuned for server-based clients, but not for mobile resident applications. As an example, these databases often donot provide automatic restarts for failed transactions, but instead serve queries from the beginning even in the face of temporary loss of connectivity. As mobile devices often face higher probability of connectivity issues, the performance and correctness of mobile apps often depend on how mobile databases are designed. Typical mobile databases require a centralized storage server to host the master copy of data, and mobile clients contain location and host specific data. With the advent and growing popularity of cloud computing, enterprise software vendors are now outsourcing data storage needs of their applications on to third party hosted commodity servers that can be accessed over the Internet. Not only this optimizes the infrastructural requirements of

enterprises, it also provides a notion of unlimited elastic storage. In addition cloud hosted database access and data storage enables distributed load-balancing techniques with guaranteed quality of service for applications when applications access data over the network. With these features cloud oriented database repositories are complimentary to traditional Mobile Databases.

We see a gap in the current technology offerings in what mobile devices need, versus what is available. It should be noted that the term *mobile devices* apply specifically to smart-phones, tablet computers or even laptop and palm-top computers that can remain active even on the go. It does not include various peer-to-peer devices that connect via bluetooth, RFID, NFC etc. As they are primarily for P2P communications, they cannot or need not talk to the cloud and hence are not included in this research.

1.3 Project Applications and Impact

Any individual or a corporate user would be able to subscribe to the MDaaS service. Upon subscription, the user will be able to download the necessary mobile apps into their devices with which they can connect to the MDaaS. The existence of MDaaS will be transparent to the mobile applications and MDaaS will provide all the requisite Quality of Service for any database transactions that are initiated by the mobile device. It is also expected that the MDaaS solution can be adopted by enterprises. A possible use case may be a corporation would be able to subscribe to the service for all its users, whose data will reside in the cloud and accessed by their users alone. This brings multi-tenancy and data privacy considerations to the design.

1.4 Project Results and Deliverables

Upon successful completion of the project a prototype for some of the key components of Mobile Data as a Service product is expected. It will be executed in two phases:

- *Phase 1:* In CMPE 295A timeframe, emphasis will be laid on identifying the use cases, formulating engineering requirements and high-level design of the MDaaS solution.
- *Phase 2:* In CMPE 295B timeframe, a prototype comprising of some of the key components of the product will be developed. The end of Phase 1 will identify the key items.

The following is a list of the expected deliverables from the project:

- By the end of Phase 1, a completed requirement and design specification of the MDaaS solution will be available. This will be Chapter 2 and 3 of this document. In addition, a high-level implementation and test-plan will be available (Chapter 4 and 5 of this document).
- The end of Phase 2 will make all infrastructure and product code along with the final design and implementation report made available.

We will also explore the possibilities of publishing the work in leading journals and papers once the design progresses and reaches maturity.

1.5 Market Research

In their latest management research, *Mobile Cloud Computing Industry Outlook Report: 2011-2016*, Visiongain examines the market and how the ecosystem players are leveraging mobile cloud solutions to enable their strategies and business models. According to the report, adoption of cloud would open rich possibility in double-digit growth of computing in the next decade – at about 16% CAGR over the period 2013-2018, with annual federal cloud computing market to hit $10 billion landmark by 2018 [51]. The US Federal Cloud Computing market forecast for 2013-2018 envisions "*mobile cloud service for the federal government as a fledgling market in the beginning of steep growth"*.

According to ABI Research [50], the *limited processing and memory capabilities of mobile devices have always required some use of the "cloud" for delivery of mobile applications and services. However, the challenges inherent in application development as well as the desire to enhance applications with location, presence, and other value-added services are driving greater use of the cloud for creating advanced mobile applications.* This serves as an interesting data-point for the current work as more and more modern mobile apps are becoming location aware and hence as they move from one location to another, their data requirement changes and they start over new downloads. This effectively translates to the capability of mobile devices to be able to talk to a new database server as they move from one region to another. The report goes on to state that *the enterprise will be the primary beneficiary of cloud services as mobilizing enterprise data and enhancing existing mobile applications and services will tremendously enhance productivity.* This effectively means that there would be requirement to support corporate users managed by a group of administrators, who would access *secured* data while on the move.

1.6 Technology Trends

Mobile Cloud Computing is a new paradigm where Cloud Computing and Mobile Compute Services overlap. Although *mobile cloud* and *cloud computing* may appear analogous, there are crucial differences in terms of platform infrastructure, security, data storage and others. Some of the key technical trends of mobile computing are as follows:

- Mobile Computing has made easier access to data than ever before. Social media sites, such as Facebook, Twitter, Flicker, LinkedIn allows sharing of large volumes of data than ever before. This has led to acceleration of *consumerization* of data and IT infrastructure [51].
- Recent advancements of cloud has enabled certain essential services like emails to hosted in the cloud. Microsoft, Google and Force.com has email services that are hosted in cloud and allows users to easily access them from anywhere in the world on users' mobile devices.
- With the advancements in the above services, corporations have recently invested heavily on security. This is a crucial aspect as most of the smartphones and tablet computes are susceptible to hacking and serious attacks. Stronger and dynamic

encryption, and various data hiding techniques have made mobile devices much secure [52].

- The speed of mobile accesses have increased manifold in the past few years with wider adoption of 4G. This has improved utilization of bandwidth and lowered latency of transactions.
- HTML5 is an important step for mobile web applications. HTML is a document publishing markup language that provides a means of specifying web page elements such as headings, text, tables, lists, and photos. Enhancements that HTML5 introduces address the need for web application support. HTML5 also allows specification of offline support, which makes local storage possible, helping with connectivity interruptions [53].
- A database enables easier access to data from applications rather than using system or library API calls to write XML or some proprietary format into disk files. With the use of databases, it is possible for applications to avoid using SQL parsers and make the applications portable across the platforms.

1.7 Literature Survey

In this section we survey the current *state of the art* of mobile databases and cloud databases.

1.7.1 Mobile Databases

We begin with a discussion of some of the crucial features and special needs that are expected from databases when in mobile environment:

- *Distributed nature of databases:* As mobile databases are typically replicated into many other client copies, this reflects a truly distributed architecture. Careful consideration needs to be made on caching and proxying needs in order to cater to cases when client databases are powered off.
- *Connectivity management features:* Databases must be able to cater to frequent connectivity loss in these kinds of environment [5].
- *Transaction management features*: apps initially interact with a local copy of the database, which sync up with the master copy at a later time when connectivity is restored.
- *Data synchronization features*: As data is stored in a distributed manner, database must have efficient conflict resolution and synchronization techniques.
- *Data security features*: Since data is stored in individual devices, adequate cryptographic measures must be adopted so that unintended users do not compromise the database integrity and security.
- *Location aware database features*: Unlike server based database access clients, most mobile databases are accessed from a variety of locations. Accordingly most database designers distribute their databases into two categories, viz. location based and location independent [4].
- *User Mobility aware database features*: Typically mobile users move around a micro geographic area throughout the day. Thus there needs to be an optimization in the database architecture for considering these factors [6].
- *Query optimization features for Mobile Accesses*: Accesses to the mobile

databases need to be optimized based on the fact that an user can initiate a transaction in one mobile location and finish in another. Thus the coherency and integrity of transaction must be guaranteed and only one copy of data should be treated as the master copy.

In a typical mobile computing environment, the register sites are responsible for managing service as well as for managing mobility. The Home Location Register (HLR) is a database, which is used for storing location information and service profile of a customer. The Visitor Location Register (VLR) is another database that is used for storing data about a user when they move to a new location. This is a temporal database containing information about visitors in a particular database for a Mobile Service Center (MSC) under a Base Station (BS). The above mobile infrastructure components could be viewed as either a two-tier or a three-tier database architecture [3]:

- *Two-tier database model*: In this model, the BS-MSC-VLR- HRL serves as a database server that contains actual information about different users, while the mobile handsets serve as a client.
- *Three-tier database model*: In this architecture, the VLR is responsible for application and data management logic and gets real time updates from both HLR and BS/MSC about user location information.

As the databases in VLR and HLR need real time updates, they require high performance, efficient concurrency control, data consistency, and replication control. This database requires reduced overall computation time, and must be able to coordinate heavy traffic in queues from mobile application clients. A client server architecture is proposed in [3] in which the Server Processor hosts a memory resident database that can perform real time concurrent updates, and a Client Processor that acts as an SQL gateway from mobile applications. Mobile Database systems consist of two key types of nodes, viz the database nodes and their reference directories [2]. This concept was extended to add an additional third type of node for improving reliability and performance through caching as discussed below [13]:

- *Database Nodes (DBN):* These nodes represent client devices with their own copies of mobile databases.
- *Data Directory Nodes (DDN):* These nodes represent devices where schemas are stored for DBNs and consist of a directory to indicate which DBN holds what data. It is also responsible for running query execution plans DD's play a central role in this infrastructure, and is responsible for synchronization, and coordinates data access by different DBN's. Typically Packet Data Serving Nodes (PSDNs) act as DDNs.
- *Caching Node (CN):* A node that holds copies of fragmented data from DBNs. It's a snapshot of data residing on different DBNs, and is used when DBNs are disconnected. Typically Base Station Controllers (BSC) serve as caching nodes.

Figure 1.1 attempts to combine the above architectures to form typical reference architecture for mobile databases. It may be seen that this architecture represents an

ecosystem of federated databases that can work in conjunction to form a large distributed repository of data.

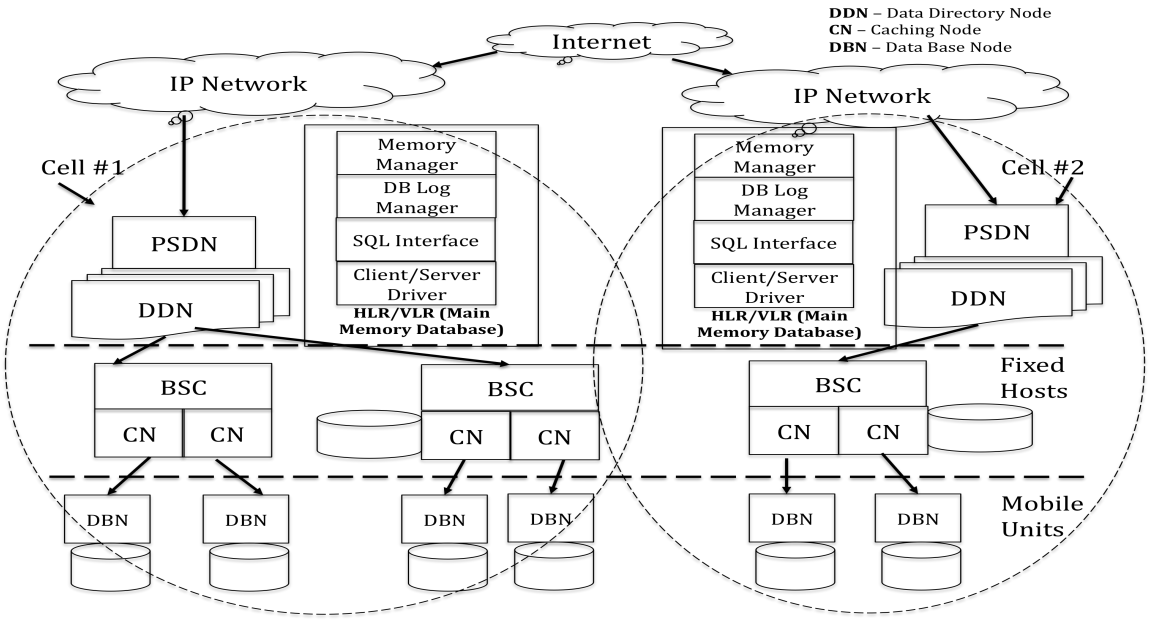


Figure 1.1: A Typical Mobile Database Architecture

This model work fairly well when one or more mobile devices are down as caching nodes can proxy them. As soon as the actual nodes come online, the data from the caching nodes are used to synchronize. A drawback of this model is its scalability consideration. It also depends on availability of network, service provider features at the master database, and limited storage capability at the mobile device nodes. If multiple nodes within a domain go offline frequently, then there will be larger number of proxy nodes. Later when these nodes come online, there will be lot of communication overhead in synchronization.

1.7.2 Cloud Databases

The main objectives and special needs of cloud databases are as follows:

- *Providing a scalable database solution*: This can be achieved by designing a database where database objects are not related to each other as in the case of RDBMS. In addition, the cloud databases should be vertically scalable.
- *Ensuring high availability of data*: Database access need to be round the clock for meeting the business needs, hence they should be accessible round the clock.
- *Allowing secured multi-tenancy*: Cloud databases must be multi- tenant and yet provide adequate data privacy. Cloud resources must be shared across all clients. Thus multi-tenant data access is a crucial consideration.
- *Ability to pay as you store*: Users of cloud databases must be billed based on the storage capacity that they provision in the cloud instead of having to make upfront investment. Cloud databases must be designed based on this consideration.

- *High Performance and low latency*: Accesses to cloud databases from any where in the world should incur uniform performance and latency.

An Internet based network databases cannot meet most of the above requirements, that is why, cloud oriented databases are becoming more popular day by day. Co-hosting and sharing is a major characteristic for cloud computing. As many customers may simultaneously access the same database, cloud providers need to often think on how to provide privacy to customer data. Force.com provides metadata driven software architecture that enables multi-tenant applications in order to cater to the scalability needs of thousands of Internet applications that they provide. The key design goals for a Cloud Database are:

- How can a multi-tenant app provide customization to standard data objects and entirely new custom data objects?
- How can tenant specific data be secure in the shared database?
- How can a tenant modify the business logic without affecting others?
- How can a tenant upgrade and roll in new features of the newer versions of the app without affecting others?

In order to meet the above requirements, the multi-tenant apps must be dynamically linked in nature, and polymorphic in order to meet the individual expectations. This has evolved into a metadata driven runtime engine where there is clear separation of compile time components (kernel), application data, information about the application itself (metadata) and also the information that specifies customizations of the app itself. These independently allow meeting the above design goals. An application framework that is hosted in Force.com (shown in Figure 1.2) has the following components:

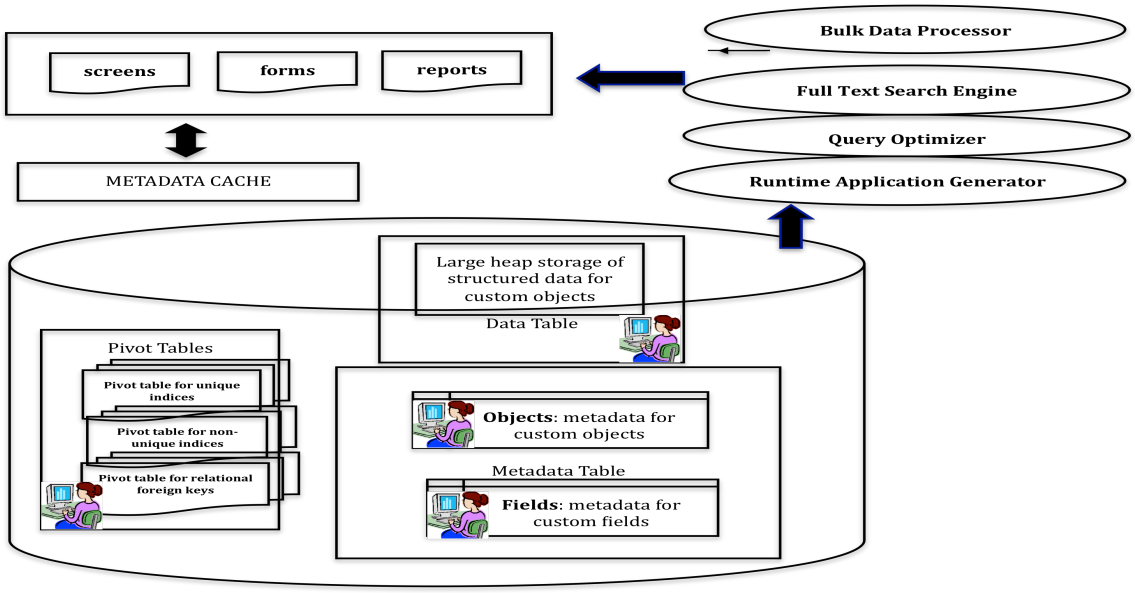


Figure 1.2: Typical Cloud Database Architecture [1]

- *Platform Runtime Engine* deals with the application and how it is executed.
- *Universal Data Dictionary* (UDD) is a central repository where all user specific

metadata is stored. This includes user forms, work flows, procedural code, tenant specific customizations, reports, and tenant specific access rights.

- *Virtual tables*: When a user creates a new application, all the above tenant specific data is stored in a metadata table, wherefrom a virtual app is generated at runtime. When someone wants to update or modify his or her customization, all that is required is a regeneration of the virtual app.

1.7.3 Mobile Cloud Databases

Mobile Cloud Computing is a new paradigm that is gaining popularity in the past few years. We examine below some of issues, needs and challenges that are faced while designing a mobile cloud database solution:

Connectivity tolerance:

- *Mobility nature*: Typically mobile devices belong to users who move around in a geographic location. It is possible that the mobile user gets into a dead-zone in between two cells, or move to locations where signal strength is less. Thus mobile databases always need to have efficient connectivity tolerance.
- *Cloud Data Center location*: Specifically in cloud computing, data centers may reside in remote locations within a geographic region. This can add to more delays in transactions.
- *Frequent disconnection*: A user in between a transaction can turn off Mobile phones forcibly. Thus a mobile cloud database needs to be tolerant to this behavior.
- *Speed*: Mobile cloud databases need to be tolerant to variable communication speed within the same session. For example, a user may start the session in WiFi and then complete it in an area covered by 3G.
- *Communication bandwidth*: Communication bandwidth depends on the number of active connections and users accessing from the same location.

Security and access control:

- *Access level security*: Accesses to cloud must be done through adequate private and public keys. Each database apps residing in mobile clients must be able to have access to their own private keys with which they authenticate into cloud.
- *Cloud security*: Cloud data storage security is an active research and investment area for data center IaaS providers.
- *Session level security*: Session level security is for implementing data security at IP layer. Most mobile OS's implement IPsec while transmission along with encryption.

Data consistency and coherency:

- *ACID compliance*: Most traditional databases require Atomicity, Consistency Integrity and Durability. Consistency may be a concern owing to frequent disconnection of mobile devices may make the clients susceptible to loss of integrity, especially when updates occur when the device is disconnected. Thus mobile cloud database architecture needs to implement a synchronization module in both client and server side

- *Backup and recovery*: The database must be backed up automatically, and balanced across different clusters. Growth and reduction of clusters must occur based on load.
- *Web services*: RESTful web APIs are needed so that mobile clients can be directed to the right data layer domain that is assigned to an enterprise at the time of signup.

1.8 Organization of the Document

With the above background, we will now embark in the process of designing the MDaaS product. The remainder of this document is organized as follows: Chapter 2 examines the use-cases followed by elucidation of the engineering requirements. A high-level design of the MDaaS solution is discussed in Chapter 3. A high-level implementation plan is available in Chapter 4, followed by the testplan in Chapter 5. Chapter 6 describes the project execution plan.

2 Requirements and Analysis

This chapter provides an analysis of the business model and discusses each aspect of it in greater detail. The analysis of the business model has been transformed into engineering functional and non-functional requirements followed by some possible use cases along with their scenario diagrams.

2.1 Business Requirements

The system has two main types of users: the regular end-user and the system administrator. The system administrators can be classified into two classes: Master DBA (belongs to the MDaaS deployment corporation) and a Tenant DBA (who manages databases of an individual tenant company deployed in the system). The regular end-users can only access (read/write) databases through the system, while the DBAs are responsible for building, removal, replication and migration of databases. It is possible that DBAs access the system through automation scripts. It is thus important for the system to support APIs that can serve those requests. The top-level Business Process Diagram of *Mobile Data as a Service (MDaaS)* is shown in Figure 2.1.1.

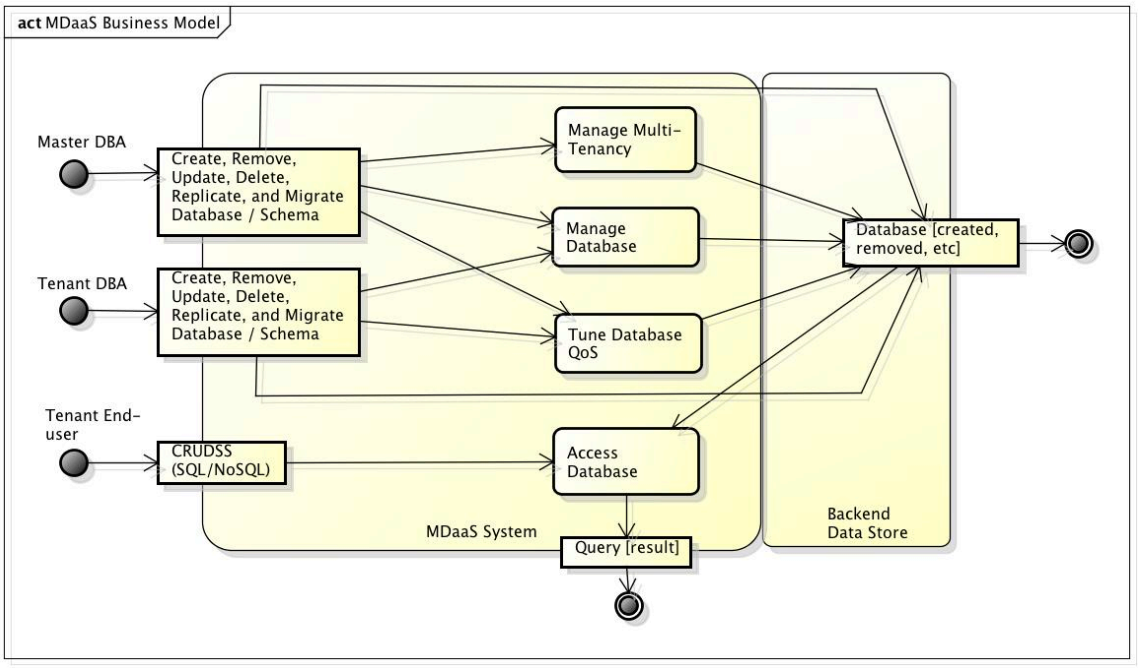


Figure 2.1.1: MDaaS Business Process Diagram

The system would create a partition in the conventional mobile database systems by segregating the data storage layer from the layer that ensures Quality of Service for mobile applications. This layer will offer resiliency of connectivity, performance tuning, database accessibility, and manage multi-tenancy for the actual databases. Figure 2.1.2 shows a decomposed view of this business model. It illustrates all the major

functionalities that are needed to implement the concept. There are some external actors like the user, their profile, data being uploaded / downloaded (database queries), connectivity database, session information etc. A user request to interact with an application database will be intercepted by a new mobile infrastructure, which will ensure Quality of Service. The QoS enabled data is utilized and processed by MDaaS to store and retrieve data from the cloud through user's mobile devices.

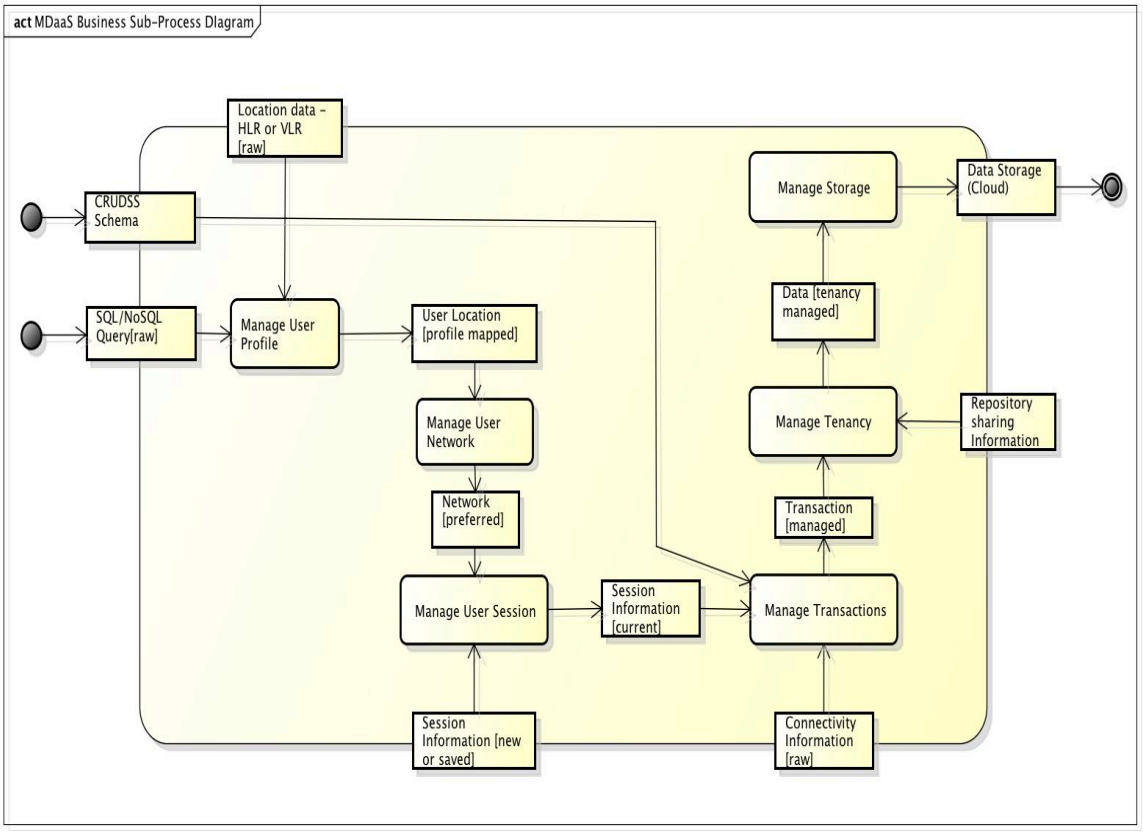


Figure 2.1.2: MDaaS Business Sub-Process Diagram

Each of the key functionalities in Figure 2.1.2 will now be further decomposed into finer logical entities in the remainder of this section and discussed in more detail.

2.1.1 Profile Management Sub-process

The *Profile Management* subprocess deals with mobile users are registered with the service. A user (or the system itself on behalf of the user) will tag their data to be specific to his/her location. These tags constitute a user's profile. A profile entry is of the form *Attribute=Value*, by means of which the user can tag their data with their preferred access location. For example, a user can tag any video and/or JPEG file to be of type *HomeType* if they access that file more from their home. Similarly any PPT or Word file could be tagged as *WorkType* if they access such files more from their work. Likewise, their favorite cuisines can be tagged as their *RestaurantType* preferences. Figure 2.1.1.1 shows the profile management sub-process.

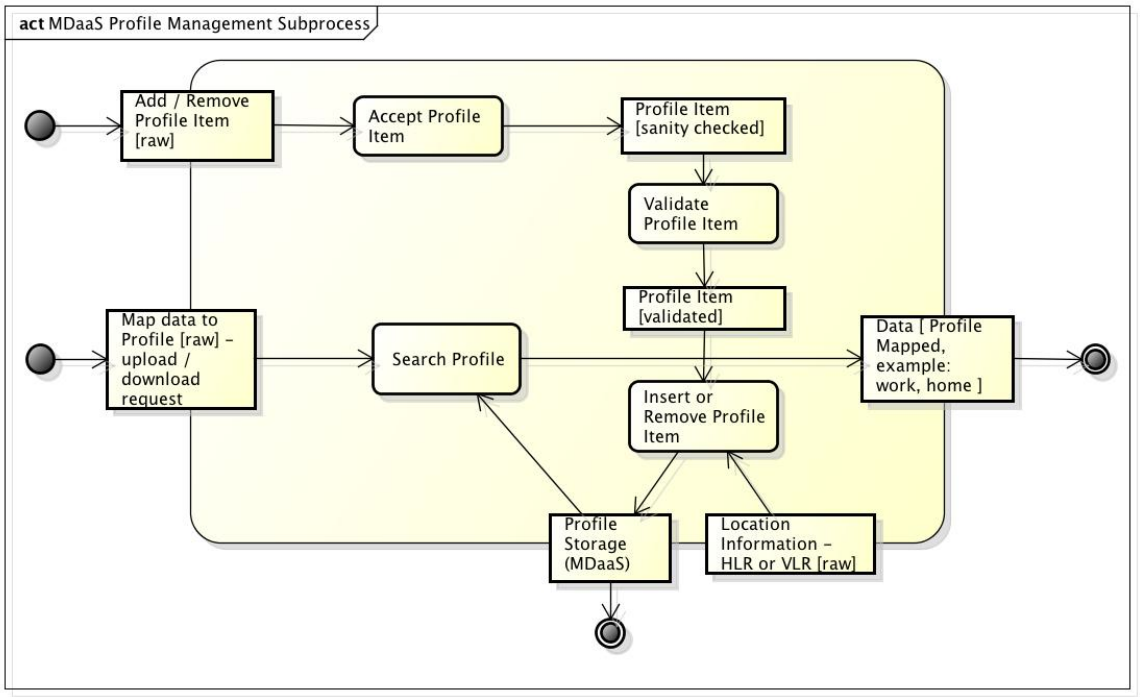


Figure 2.1.1.1: Profile Management Sub-process

2.1.2 Network Management Sub-process

Once the user has setup their profile, it will be used to setup user's preferred network and data exchanged based on their mobility. The network management component of MDaaS would use user's geo-location from HLR/VLR and their profile to setup data caches for the user. This will optimize data exchange between the mobile devices as well as help MDaaS to implement QoS at the cloud layer [5].

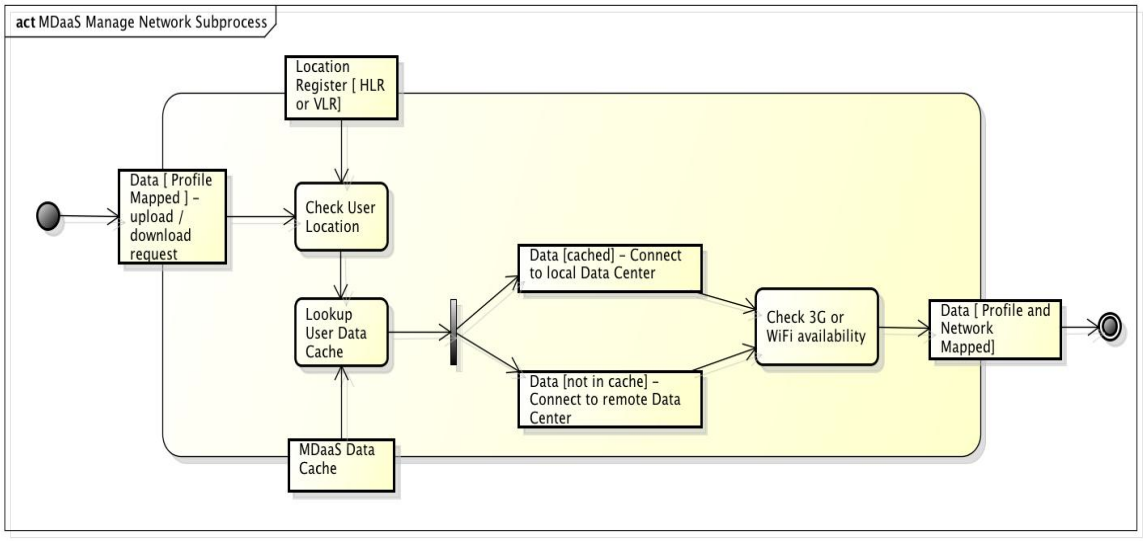


Figure 2.1.2.1: Network Management Sub-process

2.1.3 Session Management Sub-process

Once the network has been selected, the mobile client will attempt to establish a session with the master databases and storage in order to retrieve the data. Figure 2.1.3.1 illustrates the process for creation of session. One primary requirement for session management is that mobile devices can move within a region and can get disconnected from time to time. Even if such disconnection events occur in the midst of an ongoing transaction, MDaaS should be able to restore and resume previous session.

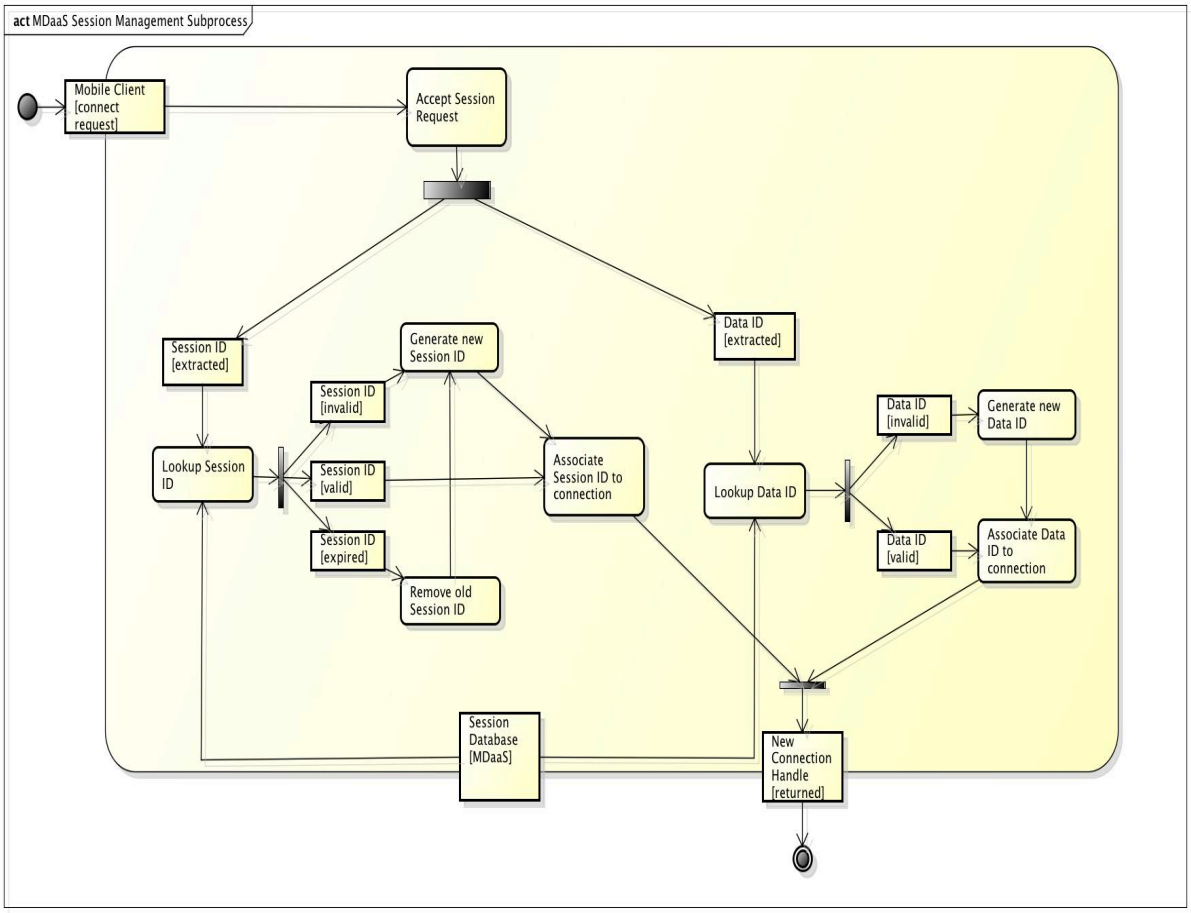


Figure 2.1.3.1: Session Management Sub-process

As seen from the above figure, MDaaS session manager will save each mobile session and tag them with the data (which could be a unique handle, viz filename, database connection etc) so that those sessions can be uniquely identified. The session information should be persistently saved in a session database that will be referred while establishing a new or resuming old sessions.

2.1.4 Transaction Management Sub-process

Transaction Management is the mechanism of ensuring atomicity of data exchange. Mobile devices, while initiating new transactions, will send a tag along with the first transaction that lets MDaaS know how many sub-transactions make up the whole exchange. MDaaS will cache intermediate transactions in its local storage and will flush data to the backend storage once all transactions are received. This will enable MDaaS to restart and resume transactions in case of interruptions. There will be a *Time-To-Live* associated with each transaction so that old transactions can make space for newer ones. A high-level overview of this process is illustrated in Figure 2.1.4.1.

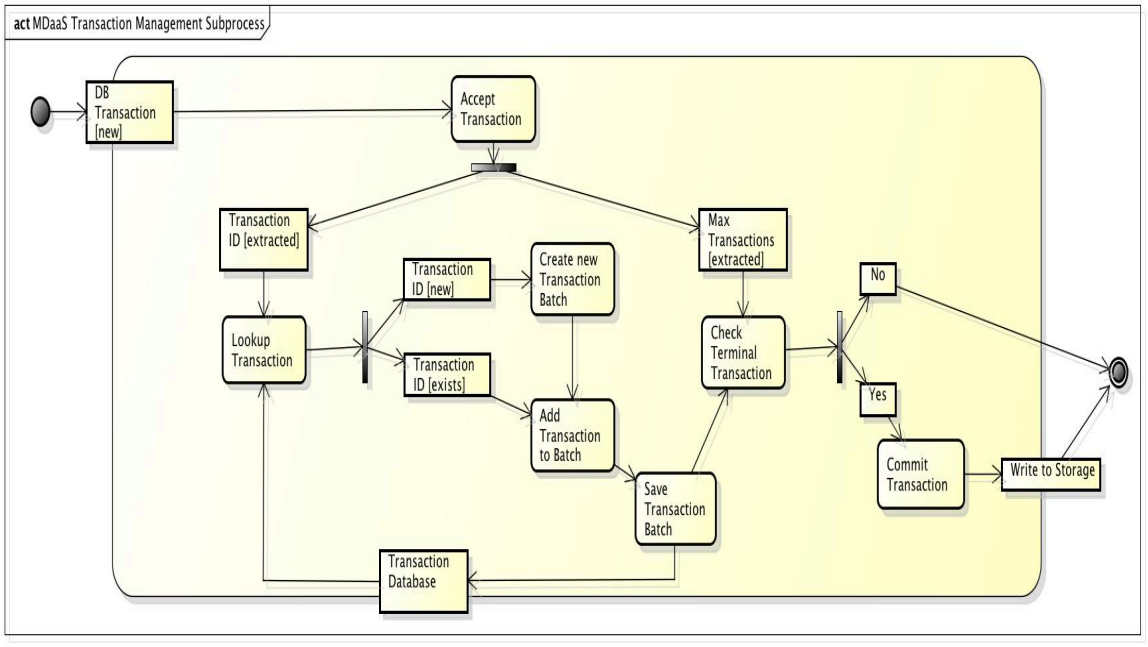


Figure 2.1.4.1: Transaction Management Sub-process

2.1.5 Multi-tenancy Management Sub-process

Multi-tenancy Management is responsible for selection and provisioning of database tenants. This module should make sure that data from two different customers (tenants) with similar schema could reside in the same table. For example: Two companies, viz. Pepsi Corporation while trying to store <EmpName,EmpID,Address,StartDate> will be able to share the same table with Coke that has a schema of <EmpName,EmpID,PrevEmployer,StartDate> if desired. Even though their schemas are mostly similar, they, with their own privacy and custom schema requirements, must be able to share the same table in order to optimize space. Typically, multi-tenant aware databases use TenantID and SecurityPolicy in the first two columns of each row in the tables to distinguish between tenants, but there may be other schemes to manage.

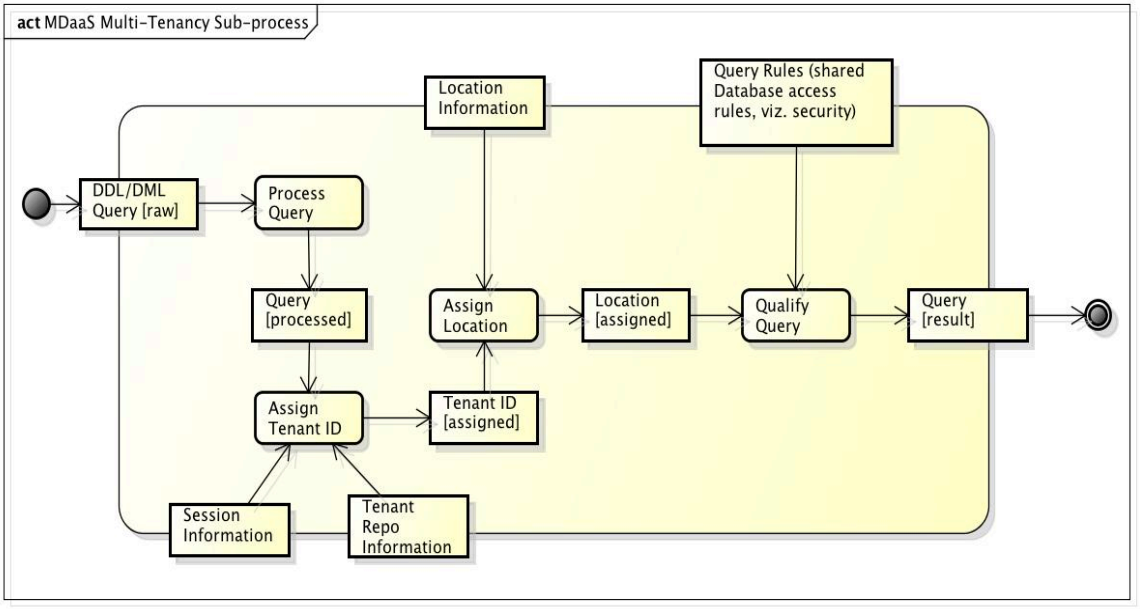


Figure 2.1.5.1: Multi-Tenancy Sub-process Diagram

2.1.6 Storage Management Sub-process

Storage Management is the final entity in MDaaS that manages how data is actually written to persistent storage. Once data is ready to be written, it is converted to the backend cloud database / storage format. Then MDaaS Storage Manager should determine the least loaded backend storage server and then establish a session with that server. Once data is successfully written (or retrieved from), MDaaS cache is updated so that subsequent reads can be addressed from MDaaS itself. This provides performance improvement to the mobile devices in not having to read data from cloud each time. A high-level illustration of this process is shown in Figure 2.1.6.1.

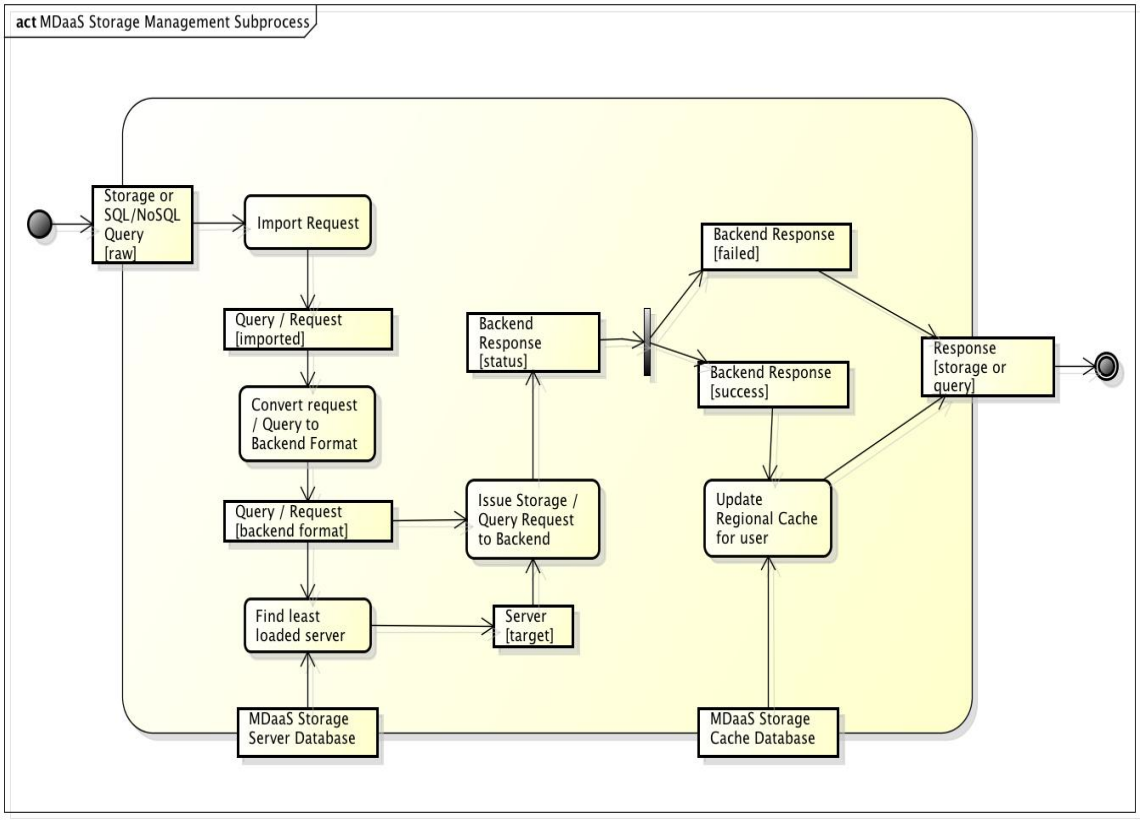


Figure 2.1.6.1: Storage Management Sub-process

2.2 Requirements Elicitation

Based on the business model and processes discussed in Section 2.1, we enumerate below the Functional and Non-functional requirements of MDaaS. These requirements will be transformed into key use cases in subsequent sections of this chapter.

2.2.1 Functional Requirements

Table 2.2.1.1: List of Functional Requirements

Requirement Type	Requirement ID	Desription
User Profile Management Service – R1.1	R1.1.1	MDaaS shall allow users to be able to tag their data into different categories (work related, home related, etc)
	R1.1.2	MDaaS shall be able to build a user's profile depending on his/her behavior: selection of restaurants/cuisine, selection of shops, books, commodities, etc. User should be able to turn on/off their profile creation.
User Mobility Management Service – R1.2	R1.2.1	MDaaS shall be able to tag data according to user's profile (work v/s home) and exchange it with the device with seamless handshake.
Searches and locality of reference – R1.3	R1.3.1	MDaaS shall be able to search for data as per user's profile and produce directed results when user is at a particular location, even on the move.
	R1.3.2	MDaaS shall be able to provide a mechanism to provide search results from user's own data and tag results in order of locality of reference.
Connection continuity of service – R1.4	R1.4.1	MDaaS shall allow users to be able to continue downloads or uploads from the point it left off. Use cases can be: manual interruptions during app installation, power/battery outage in the midst of a upload/download
	R1.4.2	MDaaS shall be able to seamlessly resume downloads instead of restarting it from the beginning even if user downloads got interrupted and user moved away from the location wherefrom they started their downloads.
Transaction	R1.6.1	MDaaS shall make sure that batch transactions are

Management – R1.6		atomic in nature. For example, if a transaction set comprises of five constituent transactions, it will succeed only if all the five are successfully committed. This shall be done in two-phase locking mechanism.
Data Management Services – R1.7	R1.7.1	MDaaS shall provide backup and restore capabilities. MDaaS shall allow user data to be automatically backed up – use of SAN/NAS techniques of auto backup to be provided - example snapshots etc.
	R1.7.2	MDaaS shall provide backup service that should be available independent of user's location and be equally performing at all times.
Data Storage Services – R1.8	R1.8.1	MDaaS shall automatically partition (shard) data and migrate data seamlessly depending on configuration.
	R1.8.2	MDaaS shall provide synchronization services across all databases.
User perspective considerations – R1.9	R1.9.1	MDaaS shall provide open/close databases depending on connection string
	R1.9.2	MDaaS shall provide indexing, selection of table capabilities etc to user applications.
Multi-tenancy – R1.10	R1.10.1	MDaas shall provide transparent access and storage of tenants' data on shared databases and shared schema space without compromising the users' data integrity and confidentiality.
	R1.10.2	MDaas shall provide an interface for DBAs set up the tenants' database schema, extend columns, monitor tenants' database performance, and perform backup and restore of user data.
	R1.10.3	MDaas shall provide an interface for System Admins to allocate storage space for tenants, perform database partitioning and data migration, monitor and optimize databases, and perform backup and restore of tenants' databases.

2.2.2 Non-Functional Requirements

Table 2.2.2.1: List of Non-Functional Requirements related to QoS

Requirement Type	Requirement ID	Description
Availability – R2.1	R2.1.1	MDaaS shall be available 24x7 99.999% of the time
Serviceability – R2.2	R2.2.1	MDaaS shall provide online technical support for customers.
Performance – R2.3	R2.3.1	MDaaS shall allow user to be able to always download from a server that is nearest to him at any point of time. This should be applicable within geographic region and/or could be an extension of REQ 1.4.2 above.
	R2.3.2	MDaaS shall optimize user data access bandwidth & performance acceleration: Cache some data in user's end-device, and also in MDaaS cloud. This will save from accessing data from actual backend.
	R2.3.3	MDaaS shall allow user access speed from mobile devices to be uniform irrespective how many user's access the systems
Interoperability – R2.4	R2.4.1	MDaaS shall work with multiple cloud-oriented databases at the backend.
Usability – R2.5	R2.5.1	
Maintainability – R2.6	R2.6.1	MDaaS shall have the ability to upgrade its own software without any interruption to customer's services.

Table 2.2.2.2: Non-Functional Requirements related to constraints

Requirement Type	Requirement ID	Description
Implementation – R3.1	R3.1.1	MDaaS shall use Java / Erlang platform that will make it to run on any platform.
Interface – R3.2	R3.2.1	MDaaS shall provide RESTful API as well as SOAP API.

Packaging – R3.3	R3.3.1	MDaaS shall be packaged in a format that can be deployed easily in a Tomcat based web server.
Compliance to standards – R3.4	R3.4.1	MDaaS shall be SOX compliance
	R3.4.2	MDaaS shall provide a means to access past and generate audit history reports.
Security – R3.5	R3.5.1	MDaaS shall stored user sensitive data in encrypted format
Legal – R3.6	R3.6.1	MDaaS shall allow selling only items that are allowed by State and Federal laws of land.

2.3 Use Case Modeling

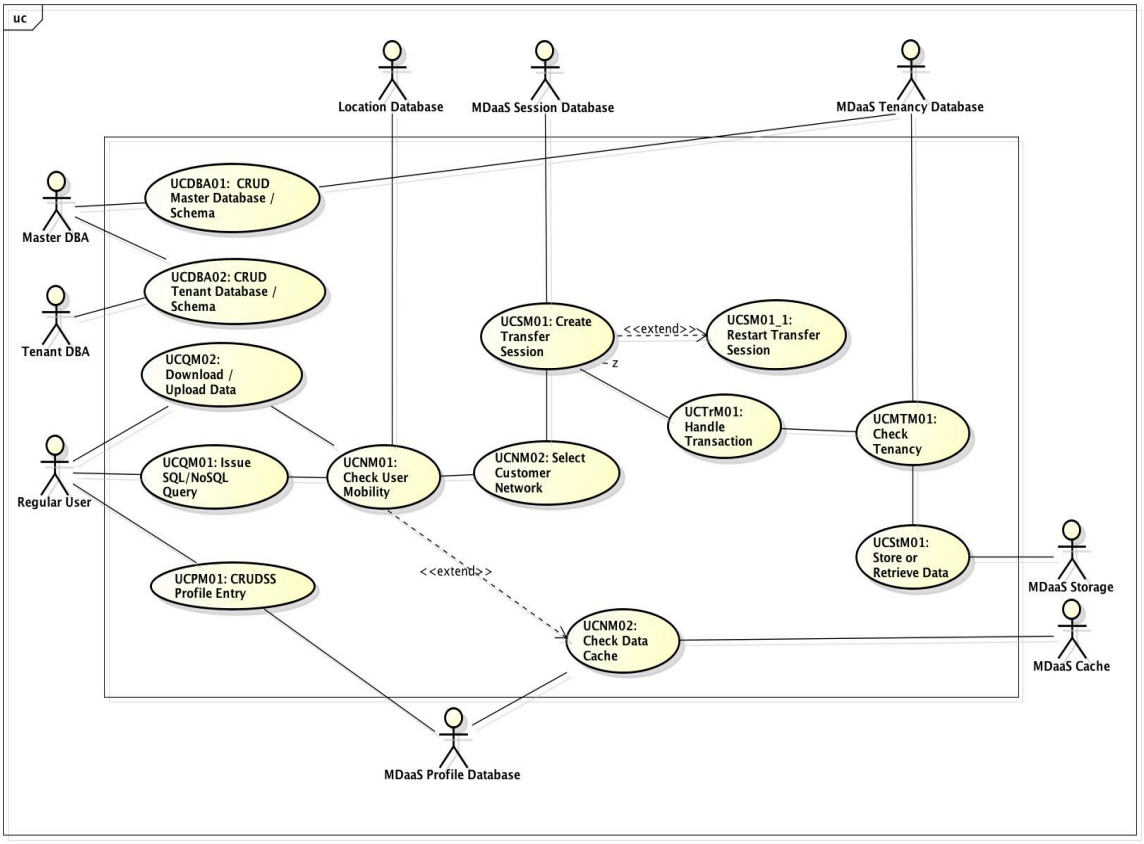


Figure 2.3.1: Top-level Use Case Diagram of MDaaS

2.3.1 Use Case Descriptions – Profile Management

Use Case ID	UCPM01
Name	CRUDSS User Profile entry
Type	Base
Actors	Mobile Client (user) MDaaS Profile Database
Description	Mobile Client wants to create, remove, update, delete, or search one or more profile entries.
Pre-Conditions	None

Post-Conditions	None	
Main Flow	**Mobile Client**	**System**
	1. Mobile client sends a request to CRUDSS an entry in the form *attribute=value*. For search/lookup requests, *value* is omitted.	
		2. Validate the *attribute*. If not a known attribute return error 3. If this is an update or create request, Validate the *value* for the specified *attribute*. If it is not an acceptable value return error 4. If this is an update or create or remove request, add, update or remove it to/from the profile database; else search the entry from the profile database. 5. Return result to mobile client.
Alternative Flow	**Condition**	**Handling**
	(none)	
Special Requirements (if any)	(none)	

2.3.2 Use Case Descriptions – Network Management

Use Case ID	UCMM01
Name	Check User Mobility
Type	Reference
Actors	Location Database Mobile client (user) System (MDaaS)

Description	System wants to determine whether location of the user has changed since last access and validate/invalidated data cache		
Pre-Conditions	Mobile client has issued some sort of query or upload/download request		
Post-Conditions	System has determined the mobility of the user by referring to the location database and validated cache.		
Main Flow	**User/Mobile Client**	**Location Database**	**System**
	1. Mobile client sends a requrest to MDaaS to upload/download data.		
			2. MDaaS contacts Location Database for user's current coordinates
		3. Looks up its database to determine geographical coordinates of the issues.	
			4. Check if there is a cache existing for the user for the coordinates [UCNM01_1] 5. Return handle to data cache if one exists, else return NULL to indicate that data must be exchanged with MDaaS.
Alternative Flow	**Condition**		**Handling**
	(none)		
Special Requirement	(none)		

s (if any)	

2.3.3 Use Case Descriptions – Session Management

ID	UCSM01	
Name	Create Transfer Session	
Type	Reference	
Actors	Mobile session manager Client Manage user session server Time System	
Description	Mobile client wants to create a session with MDaaS	
Pre-conditions	None	
Post-conditions	Mobile client gets a session id that expires after a definite time.	
Main-flow	**Mobile session manager client**	**Manage user session server**
	1. Provides user credentials in format: <Mdaas IP, username, password> format	
		2. Validate request [check] 3. If invalid request then Flag an error and return 4. Lookup session id in session id cache or session database table 5. If (session id found and session id not expired) Associate session id with connection id Else if (session id found and session id expired), Create a new session id and associate with connection

		Else Create a new session id and associate with connection 6. Return session id along with status of UCS01
	7. Store session id in the local cache.	
Alternative flow	**Condition**	**Handling**
	a. Validate request	1. Sanity check user name and password characters and length of the string 2. User name and password must meet minimum length and character combinations rule 3. If criteria for these mandatory fields succeed Return success Else Return failure
Special Requirements (if any)	None	

2.3.4 Use Case Descriptions – Transaction Management

ID	UCTrM01
Name	Handle a database transaction
Type	Reference
Actors	Mobile transaction manager Client Transaction manager server System
Description	Mobile client wants to execute set of database operations, either all of the operations or none of them in MDAAS
Pre-	Mobile client should have a session id (UCS01)

conditions		
Post-conditions	Mobile client gets execution status of the database operations it requested.	
Main-flow	**Mobile transaction manager client**	**Transaction manger server**
	1. Provides session id, number of database operations, set of database operations in JSON format: { {query_1}, {query_2}}	

		2. Validate request [check]
		3. If invalid request
		then
		Flag an error and return
		4. Lookup transaction id in transaction id cache or transaction database table
		5. If (transaction id found)
		Save Transaction to batch
		Else
		Create a new transaction id, create a transaction batch and save it to transaction batch
		6. Check terminal transaction
		If (successful)
		Then
		Commit transaction
		Else
		Rollback transaction
		7. Return the status of transaction to the client

	7. Act on transaction status.	
Alternative	**Condition**	**Handling**

flow	a. Validate request	1. Sanity check session id and query statements 2. Session id must meet minimum length criteria and query statements should not have disallowed characters. 3. If criteria for these mandatory fields succeed Return success Else Return failure
Special Requirements (if any)	None	

2.3.5 Use Case Descriptions – Multi-tenancy Management

Use Case ID	UCMTM01	
Name	Set up new tenant	
Type	Base	
Actors	System Admin MDaSS Tenant Database	
Description	Set up tenant identifier and create tenant metadata in the in the MDaSS Tenant Database.	
Pre-Conditions	Tenant has not been registered.	
Post-Conditions	A new tenant ID is generated. The tenant metadata table is created in the in the MDaSS Tenant Database.	
Main Flow	**System Admin**	**System**
	Provide tenant information	
		Read tenant's company information and tenant's supported geographical

		locations.
		Validate tenant's metatdata do not exist in the MDaSS database.
		If Tenant metadata does not exist, create new tenant metadata
		Return Tenant ID from metadata
Alternative Flow	**Condition**	**Handling**
	None	None
Special Requirements (if any)	None	

Use Case ID	UCMTM02	
Name	Define Tenant Database Schema	
Type	Base	
Actors	System Admin MDaSS Tenant Database	
Description	Define tenant tablespace and database schema.	
Pre-Conditions	New Tenant	
Post-Conditions	Tenant tablespace and database schema are defined in the shared database.	
Main Flow	**System Admin**	**System**
	Provide tenant's storage space requirements and	

	database schema .	
		Read tenant storage and database schema definition.
	Issue DDL queries to reserve tenant tablespace, define tenant tables and columns.	
		Execute DDL queries and return results
		Update tenant metadata in MDaSS database.
Alternative Flow	**Condition**	**Handling**
	None	None
Special Requirements (if any)	Backend cloud storage space is unlimited.	

Use Case ID	UCMTM03	
Name	Maintain Tenant User	
Type	Base	
Actors	Tenant DBA MDaSS Tenant Database	
Description	Add/Delete/Modify tenant user	
Pre-Conditions	None	
Post-Conditions	Tenant user is updated.	
Main Flow	**Tenant DBA**	**System**
	Issue API or submit webform to enter user	

	information	
		Read user information
		Execute API or process webform to extract opcode and user information
		If Opcode = Add, Create new entry in User table in the tenant's table space Delete all user dataupdate user information
		If Opcode = Delete, Delete user entry in the User table, Delete all corresponding user data
		If Opcode = Modify, Update user information in User table
		Update tenant metadata in MDaSS database.
Alternative Flow	**Condition**	**Handling**
	None	None
Special Requirements (if any)	None	

Use Case ID	UCMTM04
Name	CRUD user data
Type	Base
Actors	Transaction Manager Location Manager Session Manager Storage Manager MDaSS Tenant Database
Description	Read and maintain user data.

Pre-Conditions	User exists	
Post-Conditions	Return query results	
Main Flow	**Transaction Manager**	**System**
	• Execute CRUD queries on user data	
		• Read user session information
		• Map location from user session
		• Find user's tenant from Tenant Database • If user's tenant does not exist, return error.
		• Validate tenant's supported areas
		• Rewrite query with the tenant's ID and user's current geolocation
		• Send query to Storage Manager
		• Return query results
Alternative Flow	**Condition**	**Handling**
	None	None
Special Requirements (if any)	None	

2.3.6 Use Case Descriptions – Storage Management

Use Case ID	UCStM01		
Name	MDaaS Storage Management		
Type	Reference		
Actors	MDaaS storage management (System) Backend Data Cloud MDaaS cache		
Description	MDaaS wants to write or retrieve data.		
Pre-Conditions	Data has been pre-processed at different MDaaS modules and is ready for writing to remote backend cloud		
Post-Conditions	Data written and flushed to all caches		
Main Flow	**Mobile Client**	**MDaaS**	**Remote backend Cloud**
	1. Send read/write request to MDaaS		
		2. Convert data to backend format 3. Obtain handle to least loaded backend server	
			4. Query status of least loaded server and return result to MDaaS
		5. Issue read/write request to Backend Cloud	
			6. Process read/write request, respond with final status

		7. If (backend I/O status == success), then update MDaaS cache 8. Respond with status to client	
Alternative Flow	**Condition**	**Handling**	
	4(a) Backend servers busy and cannot elect a least loaded server	1. MDaaS caches data or responds to client with data from cache till cache size permits 2. MDaaS keeps a timer and flushes its cache once remote server becomes available.	
	4(b) Backend servers failed to complete read/write request.	MDaaS informs client that I/O cannot be completed	
Special Requirements (if any)	(none)		

2.4 Context and Interface Requirements

In this section we examine the scenarios and context for different components of the MDaaS product. This is modeled through *Sequence Diagrams* for the key functionalities of MDaaS. This analysis will determine how data flows across the system and timing of crucial events, which will help in determining how each subsystem interfaces with the other.

We begin with the overall high-level sequence of events that is expected from MDaaS. Figure 2.4.1 is an illustration of possible key scenarios, viz. (a) creation of user profile, (b) storing data and (c) retrieving data. As is seen form Figure 2.4.1, the user first creates their profile and specifies which data should reside in the backend cloud and what should reside in their *regional* data cache (in the mobile infrastructure cloud). Then, based on the type of data requested, request is routed to the backend or serviced from the cloud. Session and transaction semantics are then setup, which is followed by a lookup of the tenant data from the master database that resides in the backend cloud.

The remainder of the section has detailed scenario diagrams on how each action would be completed.

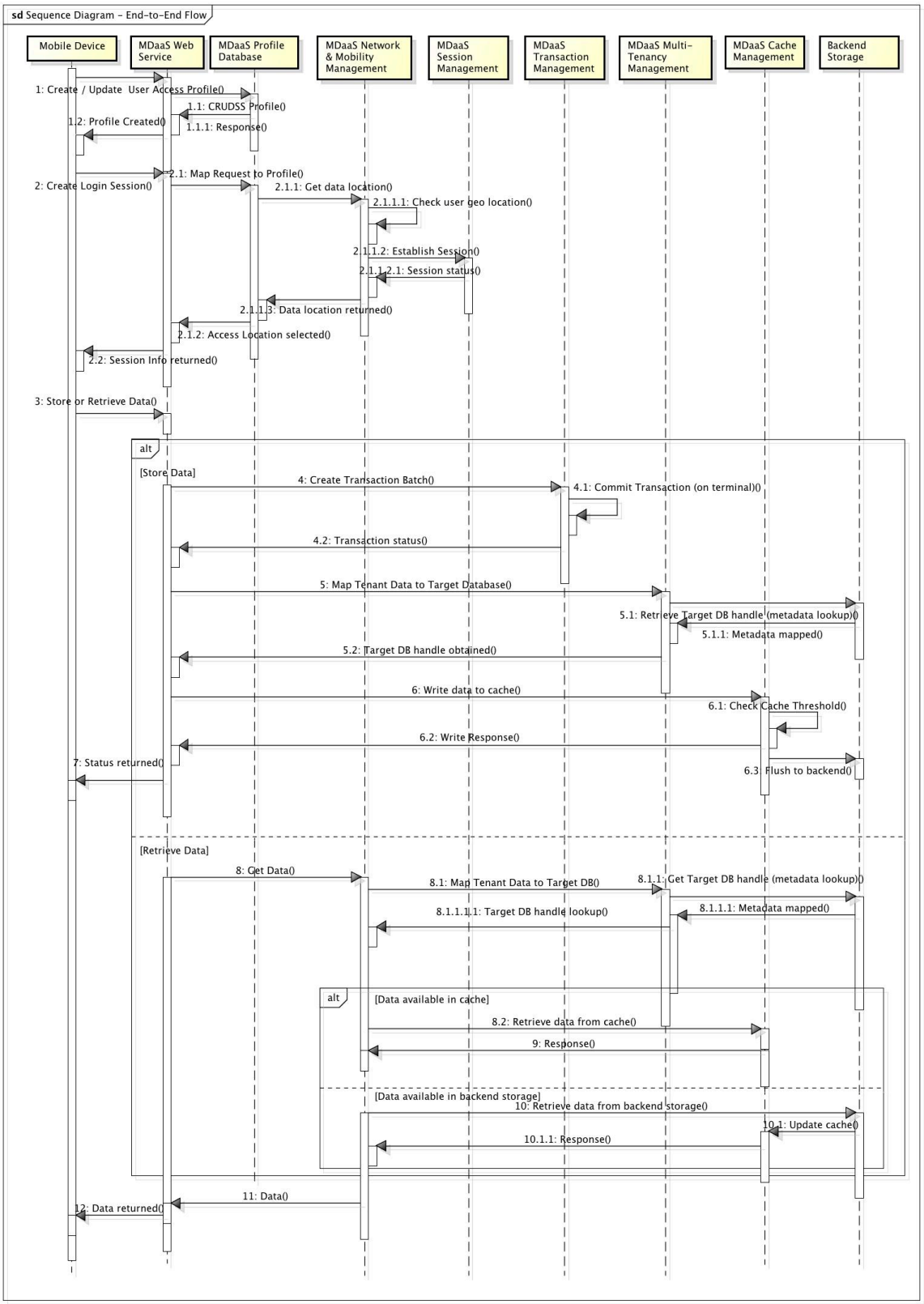


Figure 2.4.1: Sequence Diagram – End-to-End

2.4.1 Profile Management

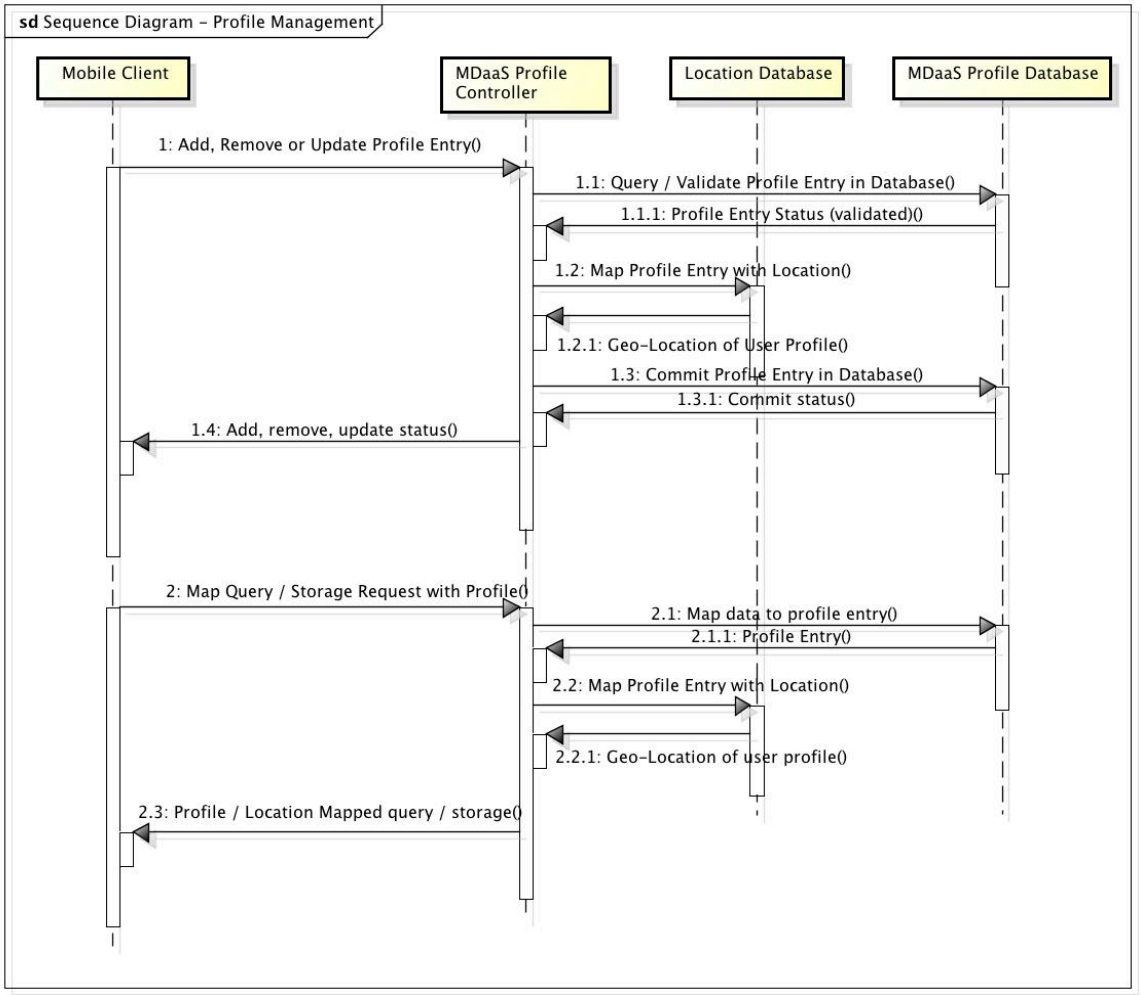


Figure 2.4.1.1: Profile Management Sequence Diagrams

2.4.2 Network Management

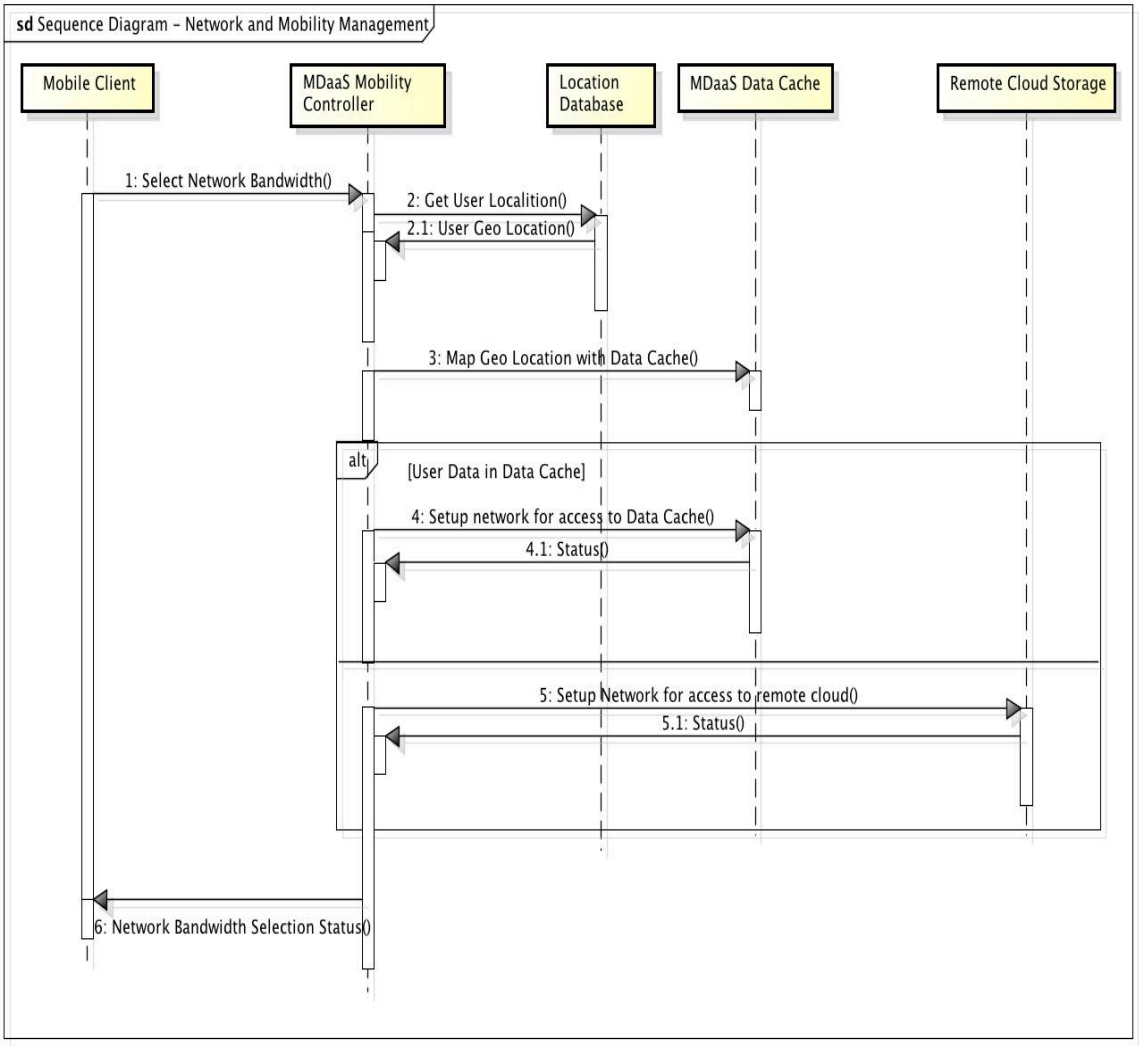


Figure 2.4.2.1: Network and Mobility Management Sequence Diagram

2.4.3 Session Management

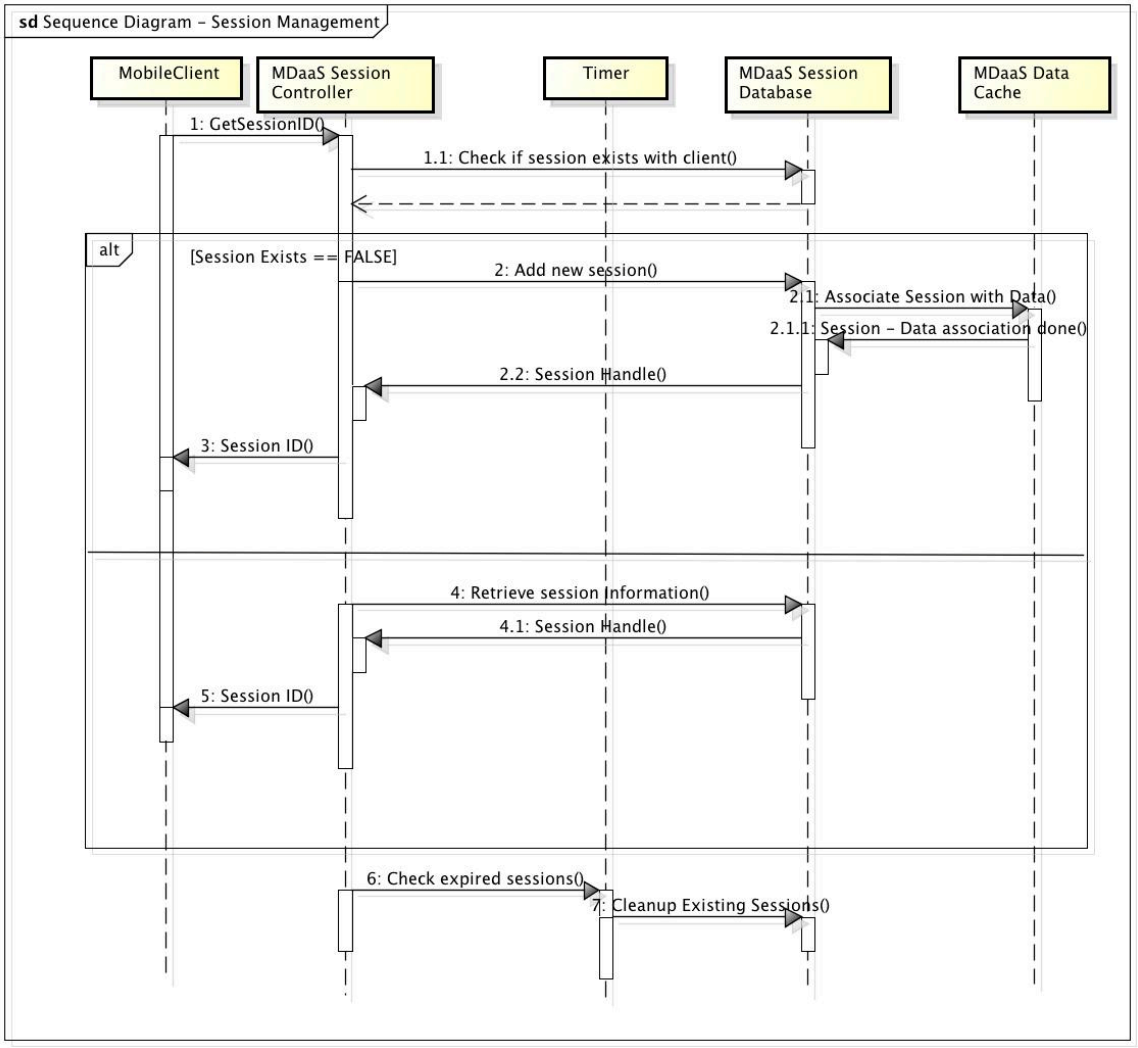


Figure 2.4.3.1: Session Management Sequence Diagram

2.4.4 Transaction Management

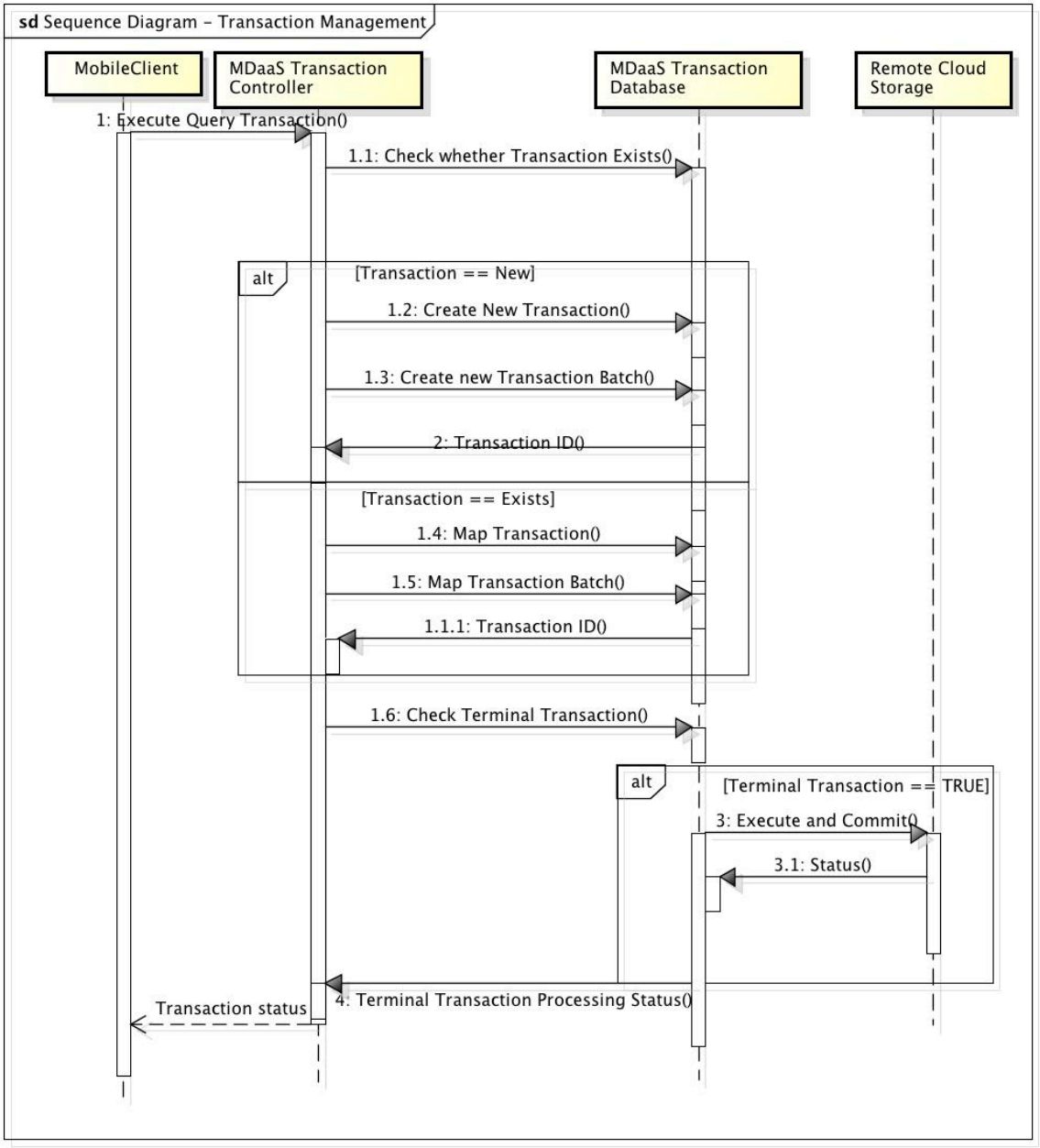


Figure 2.4.4.1: Transaction Management Sequence Diagram

2.4.5 Multi-tenancy Management

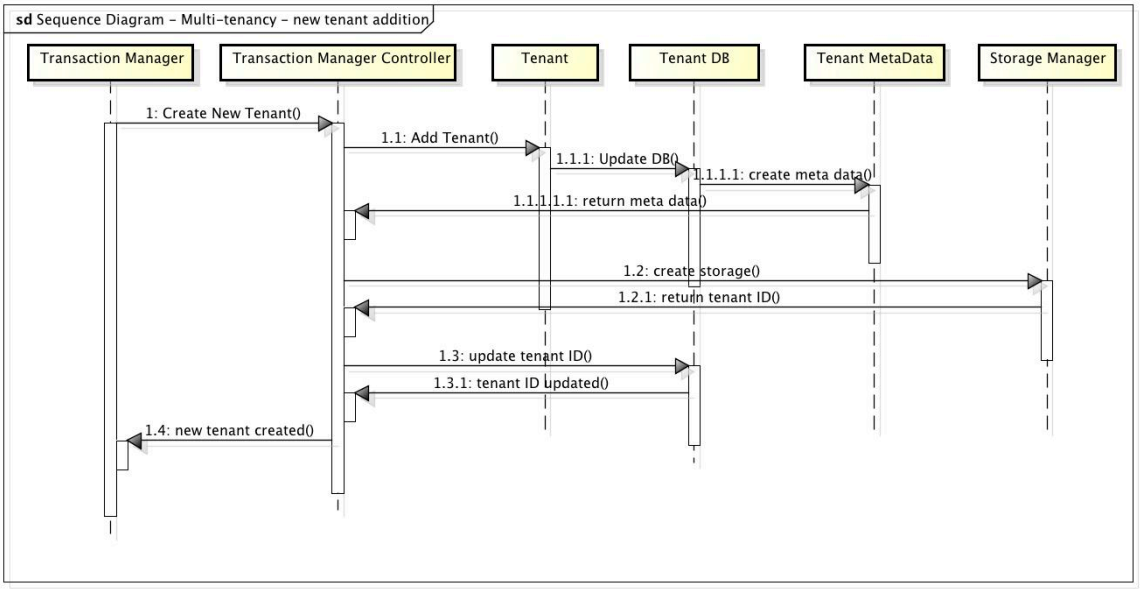


Table 2.4.5.1: Multi-tenancy Management – Addition of a new Tenant

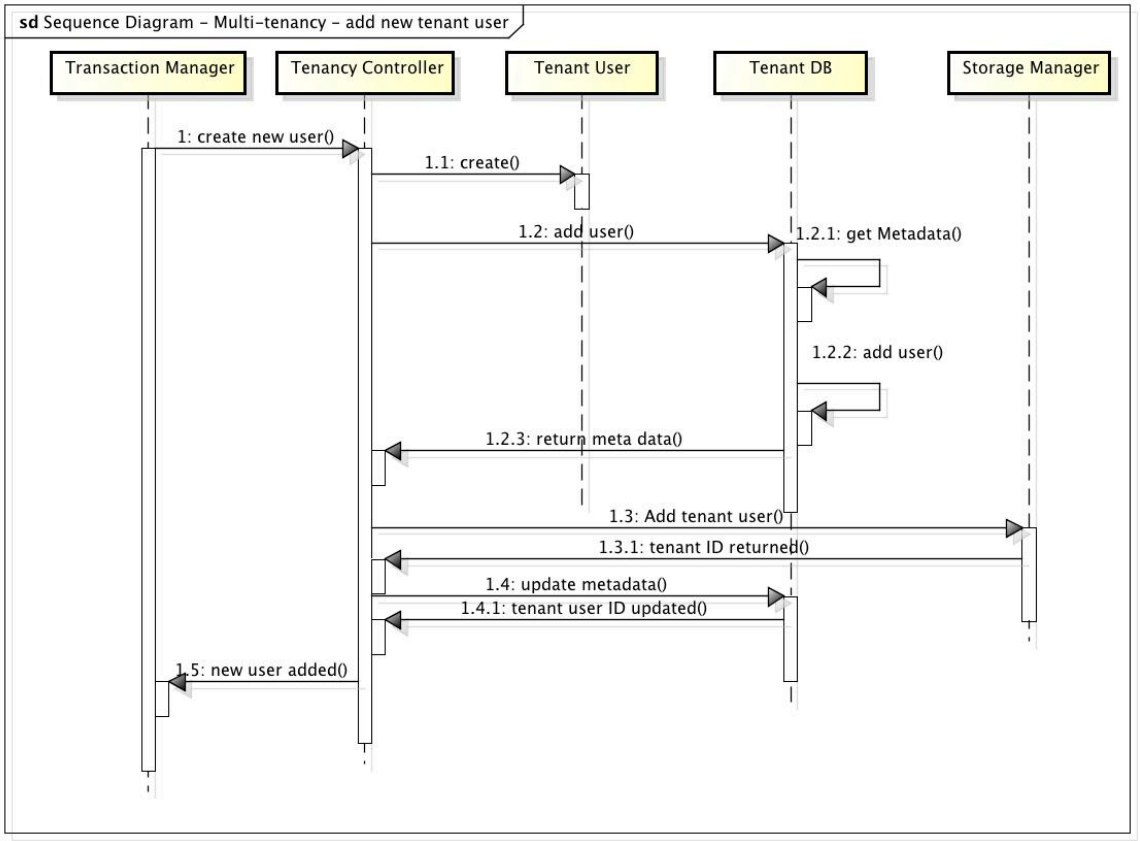


Table 2.4.5.2: Multi-tenancy Management – Addition of a new Tenant User

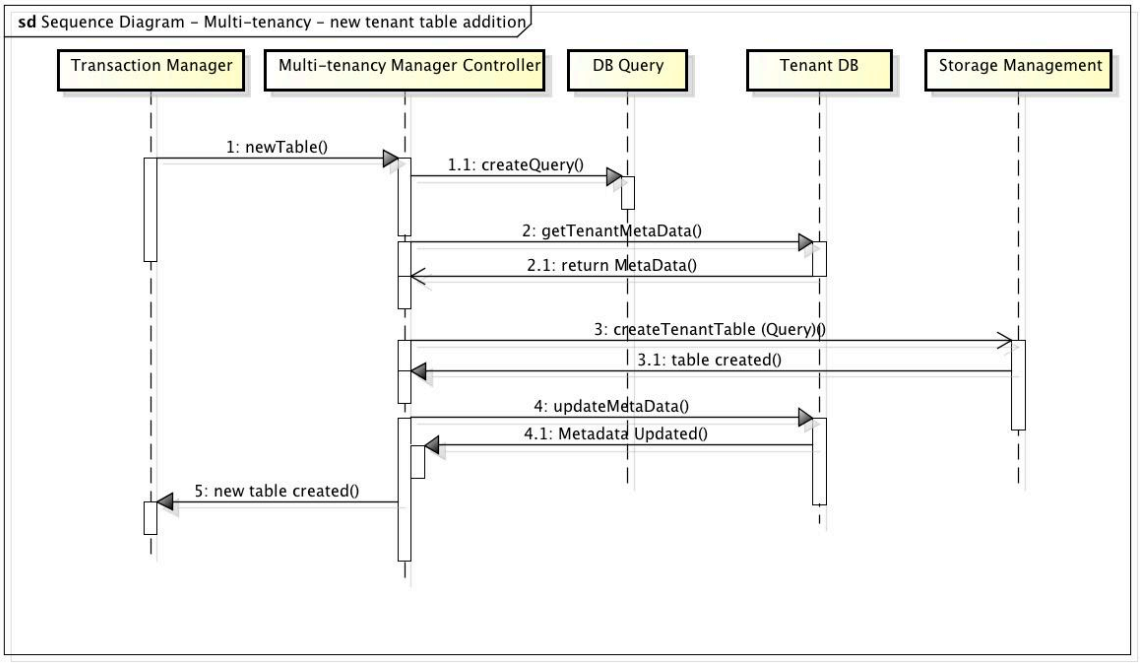


Table 2.4.5.3: Multi-tenancy Management – Addition of a new Tenant Table

2.4.6 Storage Management

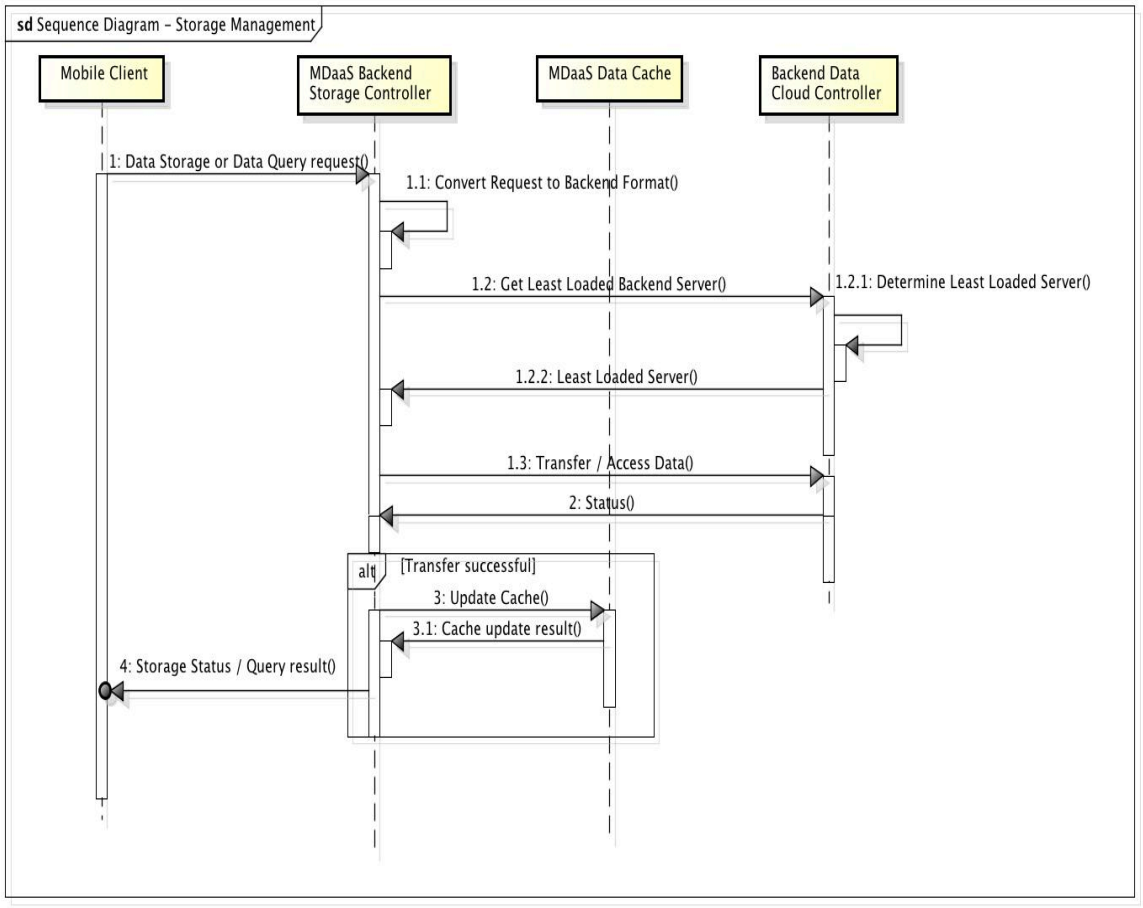


Figure 2.4.6.1: Storage Management Sequence Diagram

2.5 Resource Requirements

The following resources will be required to complete the project:

- A cloud compute infrastructure that exposes programmable APIs for remote management
- A cloud storage infrastructure that enables storing customer and control data
- Web server for REST access
- Databases: NoSQL and RDBMS
- Version controlling tools
- Build and IDE tools

2.6 Product Value Proposition

This section summarizes some of the key value propositions of the *Mobile Data as a Service* product that we believe will differentiate it with rest of the available products in the market.

Feature	Description
Data level abstraction	MDaaS will allow applications to be freed from the burden of technical programming paradigm of different kinds of databases. Instead a generic programming paradigm will be created. MDaaS will interface with the right backend database based on the user specification & content type.
Mobile data transfer latency reduction	Mobile users will not require to directly interacting with the master databases. Instead in MDaaS model, they will talk to a *Mobile Data Infrastructure Cloud* that will abstract data location. This will improve latency of transfers as well as enhance user experience.
Highly granular Multi-tenancy management	MDaaS system will enable multi-tenancy at various levels (refer Section 3.2.2.8). Most Cloud DBs provide either one of these schemes but not all. Supporting all the above schemes will enhance better utilization of cloud storage reduce *Total Cost of Ownership (TCO)* of data, and ease management and administration of data.
Mobile data transfer resiliency	Most cloud or mobile databases does not guarantee transfer control and resiliency. In MDaaS model, the *Mobile Data Infrastructure Cloud* will keep track of ongoing transfers and will be able to resume transfers.
Data filtering and user	This will allow users to tag their data with respect to their

profile	location. Such data will be cached at various *Mobile Data Infrastructure Cloud* and enable better experience while accessing data (Section 3.2.2.3)
Locality based data caching	MDaaS will provide locality based data caching that will enable users to access data based on their preset profile.
Scalability management	MDaaS will scale up and scale out through deployment of services in Virtual Machines, as well as through use of efficient load balancing techniques. This will improve user experience (Section 3.2.2.13).
Database lookup & search acceleration engine	MDaaS will implement data indexing in its caches, which will be based on user location. This will further improve data lookups and searches (Section 3.2.2.7).

2.7 Competitive Information

It is worthwhile to mention that ScaleDB [38] is a similar cloud oriented data service product that can support some of the features that our solution will provide. Figure 2.7.1 illustrates a high-level overview of the ScaleDB architecture.

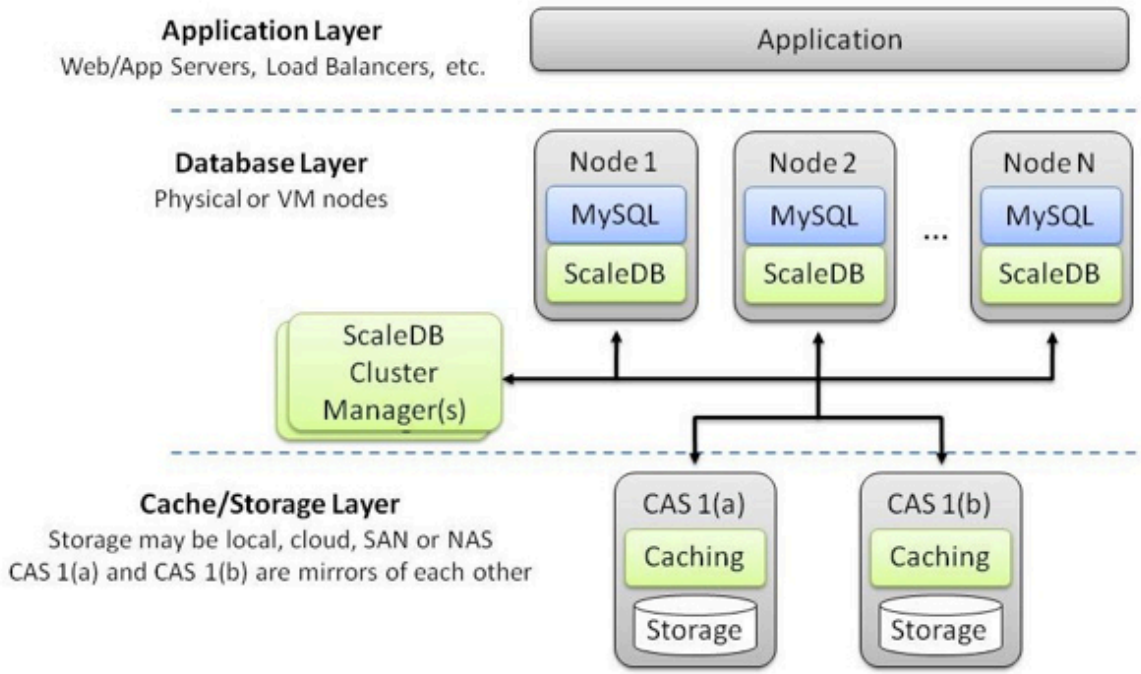


Figure 2.7.1: ScaleDB Architecture [38]

The solution is based on a number of Database Cluster nodes, which can be deployed on virtual machines, which are centrally managed by ScaleDB Cluster Manager(s). The solution is currently based on MySQL databases, which has a data-caching layer that

prevents access to databases for every data access request. We compare ScaleDB and MDaaS solution as a reference in Table 2.7.1.1

Table 2.7.1.1: ScaleDB v/s MDaaS

Feature Name	ScaleDB	MDaaS
Architecture layers	Solution is based on storing data in the cloud, but not customized for mobile access	MDaaS solution is a federation of clouds, which caches data nearer to the user location. This reduces latency of access, especially for mobile customer.
Multi-tenancy support	Even though the documentation mentions multi-tenancy, nothing is stated in public domain what levels of privacy and multi-tenancy are supported	MDaaS solution provides highly granular multi-tenancy. This improves data management and reduces TCO of data
Profile based caching	Not supported	MDaaS allows users to setup their data access profile, which is used to setup filters and caching. This helps in reducing access latency, especially important for mobile customers.
MobileDB and synchronization	Not available	Available through *Mobile Client Development Kit* (Refer Chapter 3.1.3.3)
Type of database	RDBMS (MySQL) only	A variety of DBs (Cassandra, MongoDB, MySQL, HBase)

3 System Design

This section describes the preliminary design for Mobile Data as a Service. It is based on the business model that was envisioned in Section 2.1 and uses the requirements and use case scenarios detailed thereafter. The infrastructure and architecture of MDaaS is modeled first in Section 3.1. This is followed by an enumeration of the software stack for MDaaS in Section 3.2. Subsequent sections of this chapter describe the detailed preliminary design of the major components of MDaaS.

3.1 Architecture Design

3.1.1 System Infrastructure Analysis and Modeling

The mobile end devices are assumed to be different mobile compute nodes, viz smartphones and other tablet computers that host a thin data client. Majority of the user data resides in the cloud, which each end node just consumes. An administrator user will setup data storage and databases required for end users and pre-processed in between by Mobile Data as a Service agent. Figure 3.1.1 shows the high-level infrastructure of this architecture.

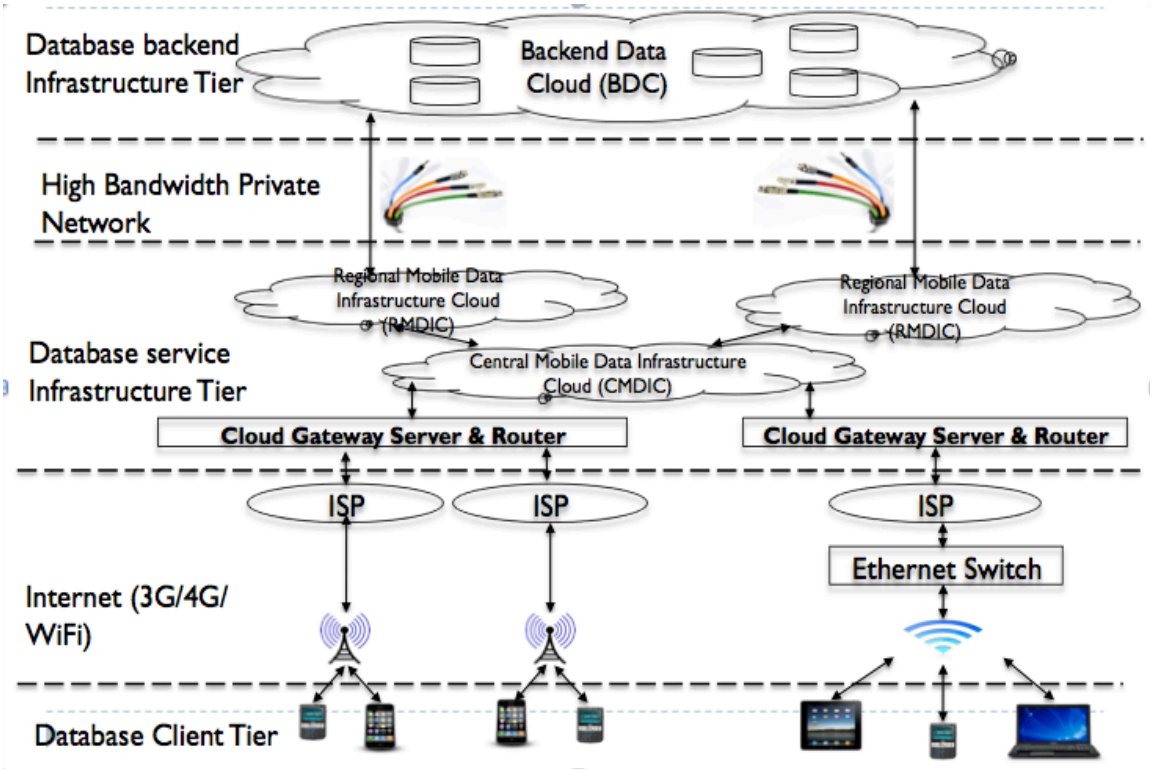


Figure 3.1.1.1: Infrastructure Diagram for MDaaS

We refer to the cloud that stores data as the *Backend Data Cloud (BDC)*. For providing services such as multi-tenancy, scalability, transaction resiliency, connectivity tolerance

etc, we propose a new cloud called *Mobile Data Infrastructure Cloud (MDIC)*. There can be multiple MDICs based on geographical regions called Regional Mobile Data Infrastructure Cloud (RMDIC). Different RMDICs are controlled centrally through a Central Mobile Data Infrastructure Cloud (CMDIC) that actually communicates with the BDC. The end client devices interact with MDICs through Cloud gateway servers and routers over wireless or WiFi network provided by the ISPs. The data is pre-processed for various QoS services in the MDICs and then transferred to the BDC through high speed (possibly wired) Internet. It depends on the corporate deploying MDaaS solution what kind of network will be used between MDIC and BDC. The performance of MDaaS is expected to be much better if used with advanced networking protocols such as fiber channel. Section 3.1.2 illustrates the software architecture of MDaaS.

3.1.2 System Architecture Context Analysis and Modeling

As discussed in Section 3.1.1, a typical MDaaS System comprises of three main software tiers:

- Backend Database Cloud (BDC) Tier
- Mobile Data Infrastructure Cloud (MDIC) Tier
- Mobile Client Tier

The main software modules in each tier are shown in Figure 3.1.2.1.

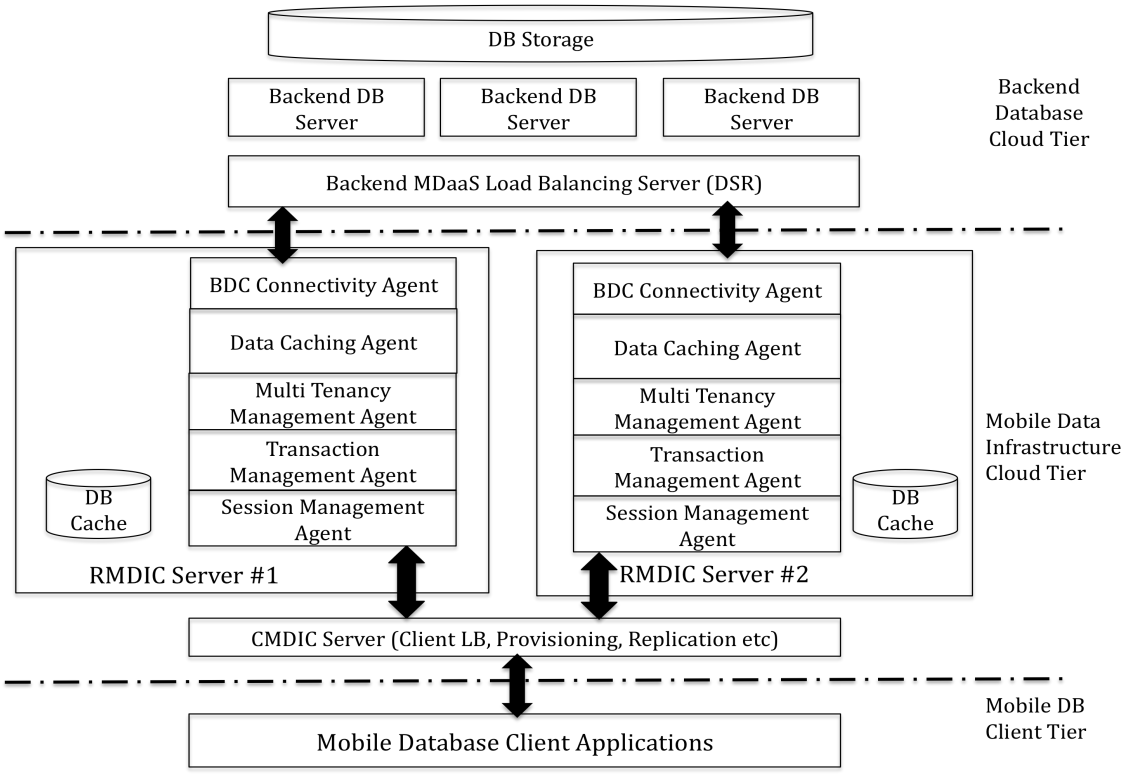


Figure 3.1.2.1: MDaaS Software Architecture Diagram

3.1.3 System Functional Modules

This section elaborates on each of the three functional modules (BDC tier, MDIC tier and Mobile Client tier) in more detail.

3.1.3.1 Backend Database Cloud (BDC)

The Backend Data Cloud hosts the cloud database and storage backend for mobile clients. Often in most implementations of cloud databases, the backend cloud storage is deployed across various geographic locations (sites). Each site can comprise of one or more database server rings. If deployed in the cloud, the end data will be automatically synchronized and replicated across rings for disaster recovery. MDaaS must provide rules and configurations when these services get triggered. Also, the BDC would require a load balancer service that can determine the least loaded storage server in each ring.

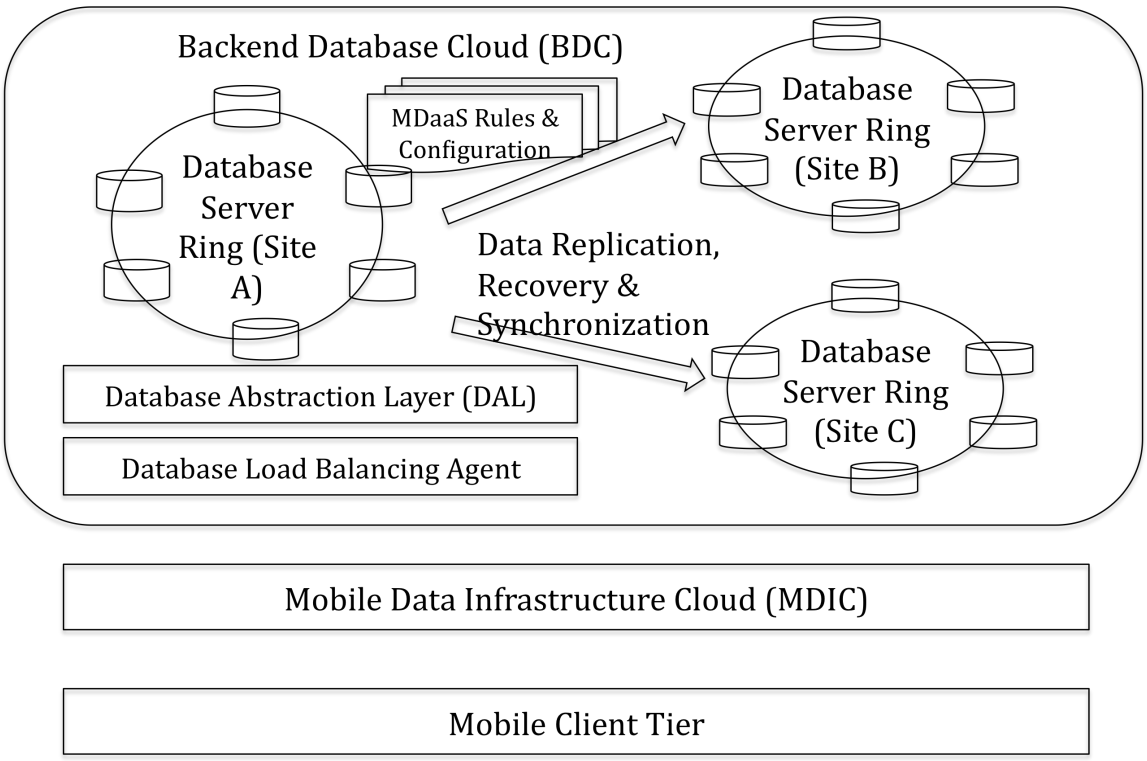


Figure 3.1.3.1.1: Backend Database Cloud (BDC) – High-level diagram

3.1.3.2 Mobile Data Infrastructure Cloud (MDIC)

The Mobile Data Infrastructure Cloud tier is the intermediate layer that provides various Quality of Service features for mobile databases. MDICs can be deployed in various data centers within a region, called Regional Mobile Data Infrastructure Cloud (RMDIC). A Central Mobile Data Infrastructure Cloud (CMDIC) is responsible for coordination

between RMDICs and also for provisioning new RMDICs. Each MDIC comprises of three main components:

- Cloud Database Connectivity Manager
- MDaaS Core Service Manager
- Mobile Client Connectivity Manager

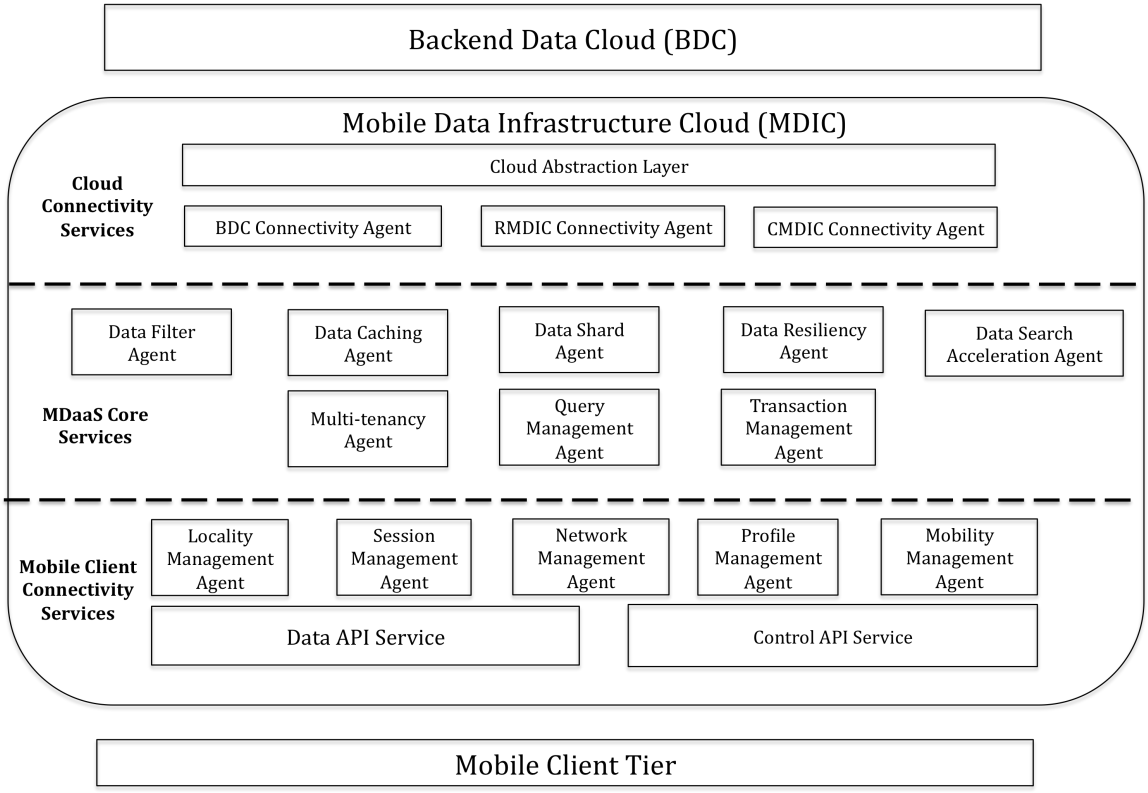


Figure 3.1.3.2.1: Mobile Data Infrastructure Cloud – High-level Diagram

We discuss below the constituent modules of the MDIC. It comprises of a number of software modules (called *Agents*) that perform their dedicated functions.

3.1.3.2.1 Cloud Database Connectivity Manager

This module is responsible for providing connectivity between the BDC and the MDIC.

- *BDC Connectivity Agent*: This agent maintains connectivity with the Backend Database Cloud. It is used as the data outlet/inlet between the MDIC and backend cloud.
- *RMDIC Connectivity Agent*: This agent maintains connectivity with the other RMDICs. It is mainly used while syncing customer data as well as control information across RMDICs (for example, when a customer moves around different regions).
- *CMDIC Connectivity Agent*: This agent is responsible for connectivity between the CMDIC and RMDICs. CMDIC connection is used when new MDICs are provisioned or for controlling MDICs.
- *Cloud Abstraction Layer*: This layer is responsible for making seamless transactions between the MDICs and the BDC. This is particularly important, as

MDICs would not require knowledge which clouds it is talking to. This provides flexibility for having different cloud vendors in the backend as well as within the domain of MDICs itself.

3.1.3.2.2 MDaaS Core

This tier implements the core services offered by the MDaaS solution. Each component is used for managing certain key functionalities of that are thought of as *special needs* unique to mobile compute nodes.

- *Database Filter Agent*: This agent is responsible for implementing filters based on which data will be accessed by different clients.
- *Data Caching Agent*: The mobility and locality management modules to optimize bandwidth will use the caching agent.
- *Data Indexing Agent*: This agent will be responsible for speeding up searches as the backend cloud databases may store data in different locations.
- *Data Search Acceleration Agent*: This agent will be responsible for parallelizing searches through use of Hadoop like techniques.
- *Database Shard Agent*: This agent will provide scalability and performance. Depending on the query and/or locality, requests can be routed to the right shard that contains the data.
- *Multi-tenancy Agent*: This module is responsible for providing multi-tenancy support for mobile database accesses. This includes, but not limited to, data access privacy, schema based storage of multiple user data in the same table, etc [1].
- *Transaction Management Agent*: This module is responsible for providing ACID capabilities to database transactions, which is missing from most cloud-oriented databases. This capability will provide mobile users to ensure reliability of batch transactions [8].

3.1.3.2.3 Mobile Client Connectivity Manager

This tier is responsible for maintaining connectivity between the cloud and the mobile clients. It comprises of two basic services:

- *Mobility Management Agent*: A crucial consideration of mobile data access is end device mobility. We envision that a user's mobility governs the amount of data that should be exchanged between the mobile client and the master database. For example, a user should be able to setup his profile for accessing work and home related content that should not only benefit the user with regard to their data usage, but also should provide savings in terms of overall network bandwidth [5].
- *Locality Management Agent*: Location based data services are a common requirement for mobile databases. The locality manager will be responsible for seamlessly setting and provide access to local regional data to the user [10].
- *Database Load Balancing Service*: This module is responsible for balancing loads for incoming database access requests. Depending on the load on a particular database, it spins up or relinquishes resources. In essence it is responsible for scaling the database service.
- *Data and Control API Service*: This subsystem will be responsible for exposing Application Programming Interfaces (APIs) to mobile clients. A few examples of

these APIs can be for offloading compute intensive tasks that depend on certain types of data, or for performing compression for WAN acceleration, or for setting up caching and filtering etc.

3.1.3.3 Mobile Client Tier

The Mobile Client Tier is implemented in the end devices. This software will be installed on the end devices as a package called *MDaaS Client Development Kit (MCDK)* during service initiation. Figure 3.1.3.3.1 illustrates a high level diagram of the software architecture in the mobile clients.

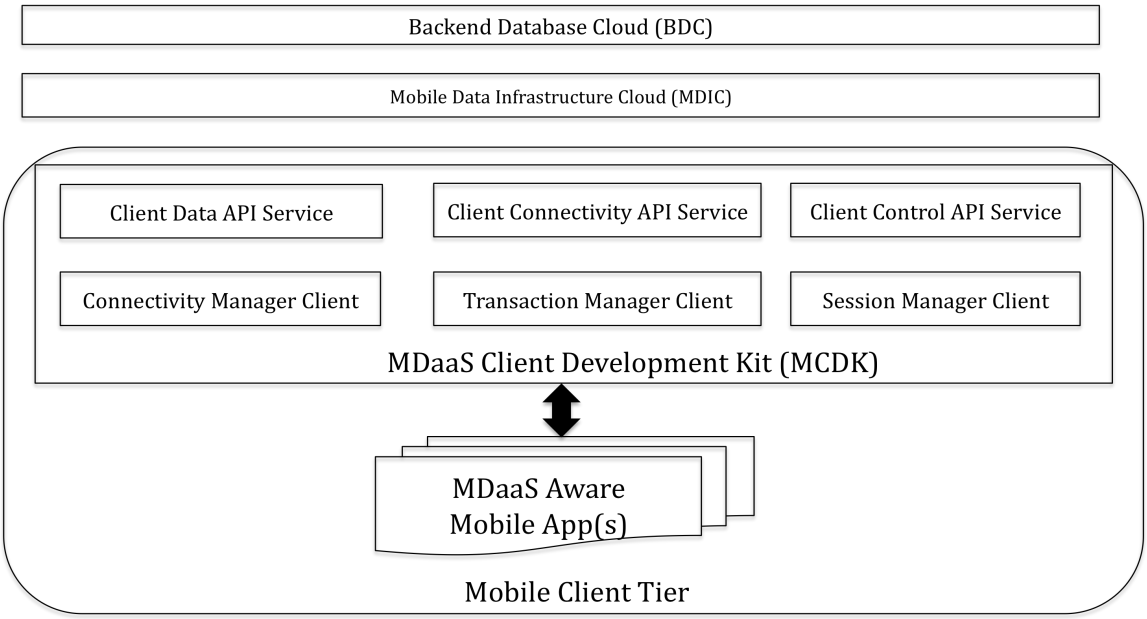


Figure 3.1.3.3.1: Mobile Client Tier – High-level architecture

- Data and Client API Services – the APIs exposed by MDaaS must be used by the mobile applications when they interact with MDaaS. This will enable extensibility and scalability of databases.
- Session Manager Client – this module will keep track of sessions between the MDaaS and the mobile client.
- Connectivity Manager Client – helps in maintaining seamless connectivity between the mobile host and cloud by choosing between the right network based on location (3G v/s WiFi)

Section 3.2 provides a preliminary class diagram and logical design of different components. A combined big picture class diagram of the whole system is available in Section 3.2.4.

3.2 System Logic and Partition Design

In the earlier section we identified *Backend Data Cloud, Mobile Data Infrastructure Cloud* and *Mobile Clients* as the three key components of MDaaS. This section describes the detailed design of these components. In this section each of these components will be examined at class level along with their main functionalities. We will also examine how users will directly talk to the MDIC and how MDIC will communicate with the backend cloud. State transition diagrams and data flow diagrams will be added to ensure reliability, wherever communications are involved or there are interactions with client devices.

3.2.1 Backend Data Cloud – BDC

The Backend Data Cloud (acronym BDC) is the actual backend data storage for the databases that are supported by MDaaS solution. In this section we examine various design considerations of the BDC. Section 3.2.1.1 discusses how load on BDC web service will be balanced using Zookeeper services. The choice of databases that will be supported by BDC at the backend is discussed in Section 3.2.1.2.

3.2.1.1 BDC Load Balancer Agent

In a distributed setup, coordination is crucial as there are lots of distributed servers, which runs loosely coupled applications that operate on huge volumes of customer data. Some specific needs for coordination are:

- Leader election for load balancing
- Group membership
- Dynamic reconfiguration
- Public subscribe
- Mutual Exclusion

Leading coordination services are: Google Chubby, Microsoft Centrifuge and Zookeeper. We plan to use Zookeeper [58] as BDC load balancer as it is a leading coordination service that is open source, started at Apache (2008), and adopted by Yahoo!, VMware, LinkedIn, Twitter, Facebook, UBS, Netflix, Goldman Sachs, Cloudera, MapR.

3.2.1.1.1 Data Model and API

For using Zookeeper, BDC will maintain a tree of data nodes called *Znode.* These nodes will arranged in hierarchical fashion, like a filesystem. Each znode<data, version, creation flags, children> will be coordinated by a *leader* Znode (/).

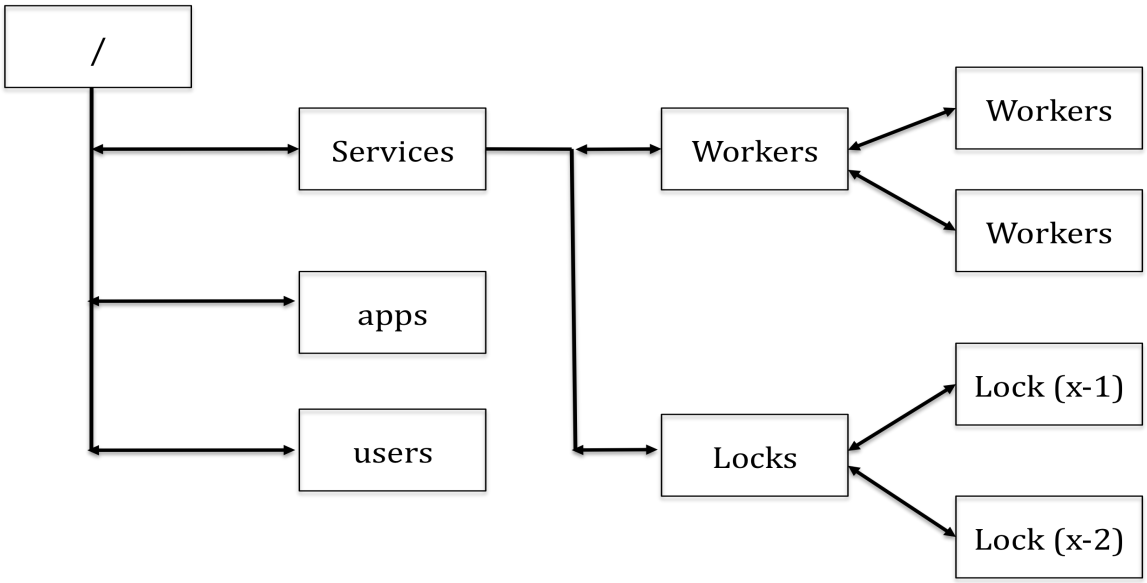


Figure 3.2.1.1.1: BDC LB (ZK) data model

The following Zookeeper APIs will be used at the BDC to provision the data nodes.

- Create znodes – create – persistent, sequential and ephemeral (remains as long as its active, disappears afterwards)
- Read/write znodes – getData & setData
- Retrieve children of znode – getChildren
- Check if znode exists – exists (znode)
- Delete znodes – delete

In BDC, all distributed database servers maintain a copy of the Znode table in their memory. A *leader* is elected at the beginning. All read requests are directed to the *follower* servers, while the leader delegates the updates. Update responses are sent back when the servers arrive at a quorum after making the change persistent (on disk). The client software that is installed on different RMDIC servers has a special ZK Client library, which knows how to talk to follower servers for reads, and send the leader server for the updates.

3.2.1.1.2 Session Semantics

- At the initiation of a new session, the ZK client library sends a prefix of operations through a session.
- If the session gets disconnected before session expires, the ZK client library attempts to contact another server. Server must have seen a transaction ID at large as the session.
- The leader generates idempotent transactions
    - Need a reliable broadcast protocol with total order – every server should commit the same series of transactions – this way, the replicas never diverge.
- Read operation
    - Sent directly to a follower server, which processes the request locally.
- Write operation

- Sent to a follower. The follower contacts the leader, which replicates the operation on a majority of servers. This method of coordination is called *Ensembling* the request.
- Examples:
  - When data is written
  - Znode created or deleted
  - Children created
  - Clients can subscribe for notification when something changes or read – Watch a value, set during a read.

- Locking:
  - Create /lock, ephemeral
  - If success, return as lock holder
  - getData /lock, set watch=true
  - Handling high load spikes, when large number of clients wake up simultaneously

```
Server: while servicing Client #1
        Z = create "/C-" sequential, ephemeral
        getChildren("/");
        if Z has lowest ID, return with lock
Server: while servicing simultaneous clients (too many requests)
        getData(prev_node), set watch prev_node
        wait for prev_node to go away
```

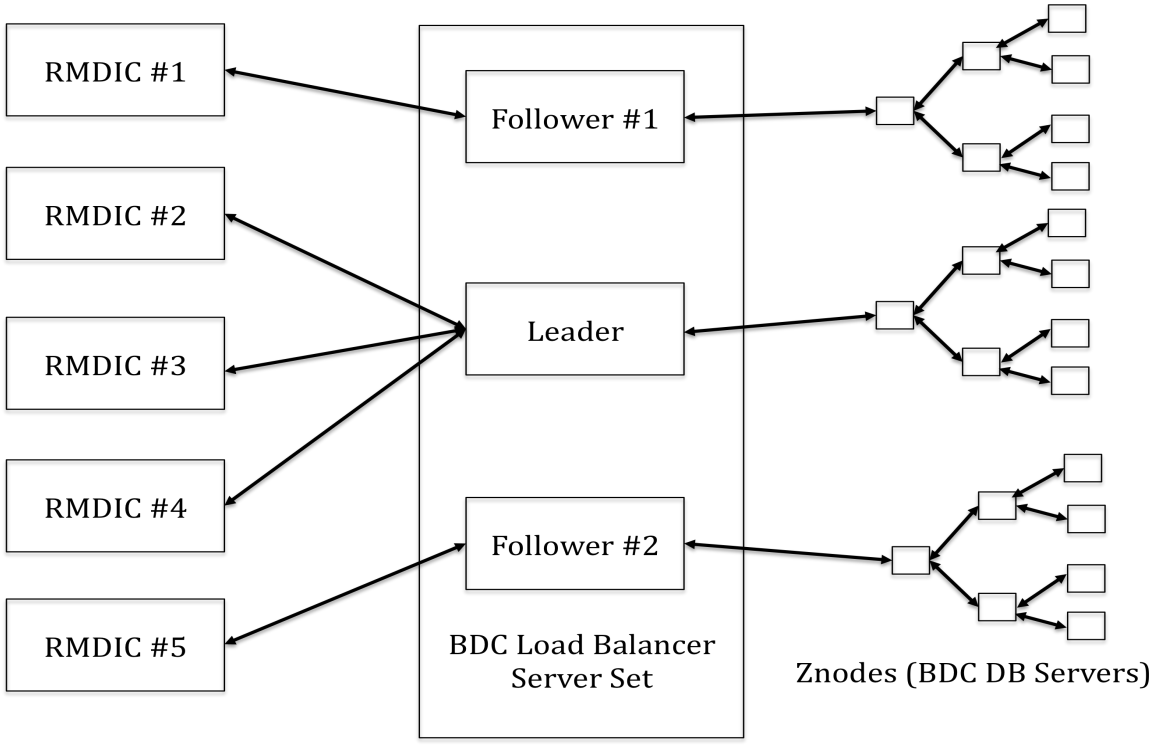


Figure 3.2.1.1.2: RMDIC – BDC LB session semantics

3.2.1.2 Databases supported in BDC

The choice of databases that are supported in the BDC is a crucial aspect of the MDaaS design, as it will be directly linked to scalability and compatibility of the solution with other services. It should be noted here that mobile client applications would be agnostic to whatever databases are supported at the BDC as the MDIC (discussed in Chapter 3.2.2) will abstract the APIs and expose MDaaS specific APIs. In this section we first examine the features of different cloud oriented database solutions and then provide a recommendation on what can be finally supported at BDC.

3.2.1.2.1 NoSQL v/s SQL databases and MDaaS

Attributes		Cloud NoSQL Database	Cloud SQL Database
Schema	Metadata	Not supported by most vendors	Required for multi-tenancy requirements
	Normalization	Required for multi-tenancy requirements	Supported
	Indexing	Not supported	Not supported by most vendors
	Multi-tenancy	Supported	Supported
	Data organization	Related data organized as collections as doc, graph and key-value pair.	Related data organized as rows in the form of a table
Performance and scalability	Migration	Easy to migrate data	Easy to migrate data
	Sharding	Supported	Not supported by default
	Replication / Splitting	Supported	Supported
	Vertical and horizontal	Not guaranteed	Scalable in both vertically and horizontally
Transaction Management	Atomicity	Eventual	Guaranteed
	Consistency	Not supported	Guaranteed
	Isolation	Guaranteed	Guaranteed
	Durability	Not standardized	Guaranteed

Query Processing	Language	RESTful / SOAP	SQL
	Connectivity	Low	RESTful / SOAP
Quality of Service	Reliability	High	High
	Availability	Desired	High
	Scalability	Horizontal	Vertical
	Security	High	In built
	Performance	Low	High
	Latency	High	Depends on type of query
	Delivery	Semi-automatic	Automatic
Service	Deployment	Hosted in persistent storage of cloud vendor	Hosted in specially designed RDBMS capable persistent storage, viz. Amazon SimpleDB
	Installation	Installed in Virtual Machines in Cloud	Installed in Virtual Machines in Cloud
	Billing	Pay-as-you-go	Pay-as-you-go

3.2.1.2.2 NoSQL database comparative analysis and MDaaS

Attributes	**Amazon DynamoDB**	**Cassandra DB**	**MongoDB**	**CouchDB**
Data Model	Key-value store	Key-value store	Schema-less and holds a set of collections, JSON like syntax	Organized as docs, accessed and stored as JSON objects
Query Model	2 APIs: get(key), put(key, context, value)	Thrift / Hector APIs required on top of Java SDK	Java, Python APIs	MapReduce functionality along with HTTP query APIs
Sharding	Node divided into virtual nodes with random position in	Supports clustering & data partitioning techniques for	Supports automated sharding architecture	Has no built-in sharding mechanism yet.

	key-space	sharding.		
Replication	Uses MVCC for sync between the replicas	Replication is very well advanced with ability to set replication factors & strategy.	No MVCC and no optimistic replication	Uses optimistic replication for both client and server side
Consistency	Customizable R+W > N-strong consistency, R+W < N-availability	Cassandra provides consistency when R + W > N (read replica count + write replica count > replication factor).	Provides only eventual consistency	Customizable: master-master eventual, and master-slave strong
Fault Tolerance	Sloppy quorum and hinted handoff for temporary failures	Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime.	Replication pair: if one node fails the other overtakes complete workload till the original is back	Provides replication mechanism to build a robust infrastructure.

3.2.1.2.3 Recommendations for MDaaS

Both Cassandra and MongoDB are popular Cloud based NoSQL database solutions that can be used as backend databases in the Backend Data Cloud. Cassandra provides more resiliency features as it supports automatic replication and sharding techniques. MongoDB on the other hand supports a number of programming languages, is simpler to manage and install. In the first release, MongoDB will be supported as the backend, and in future releases, MDaaS will use Cassandra for better fault tolerance, resiliency and replication of data.

3.2.1.3 Database Abstraction Layer

Data Abstraction Layer (DAL) has been created in order to keep the design of upper layer components agnostic of the internals of database access. DAL offers standard database independent API's, viz. initialize, create, read, write, and similar ones, which are called by upper layer through a special *DAL handle*, and the infrastructure takes care of routing the data to the appropriate database driver. A high level overview of DAL architecture is shown in Figure 3.2.1.3.1. The remainder of this section describes different API's that are exposed by DAL.

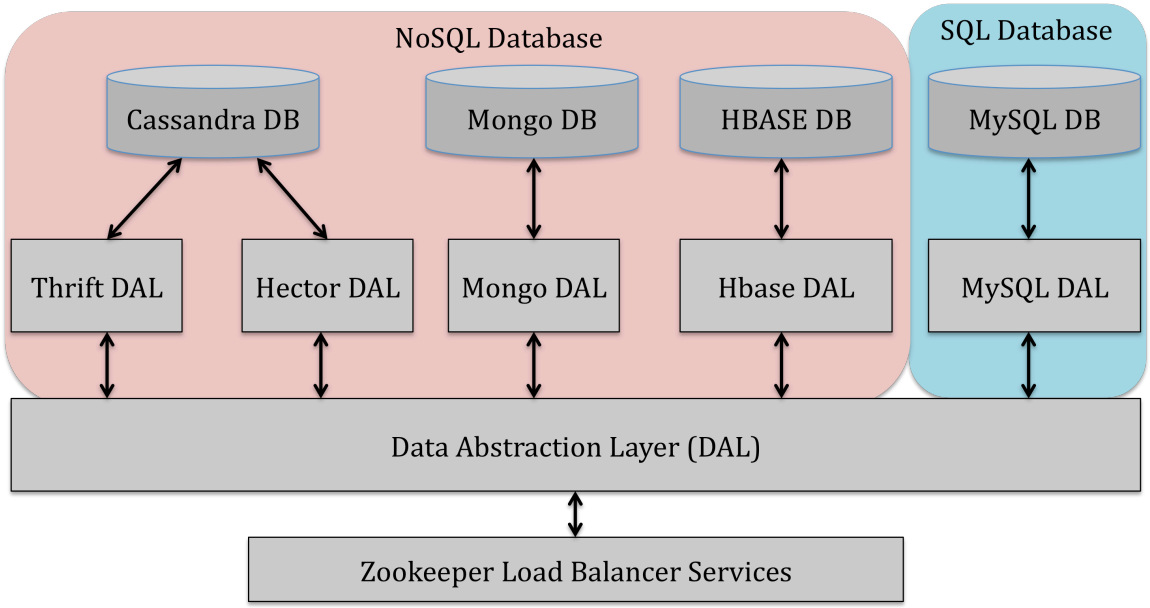


Figure 3.2.1.3.1: Database Abstraction Layer (DAL)

The intricacies of access to the database have been moved to the sub-modules (classes) implementing the abstract DAL module (interface). This enables seamlessly changing the application to use different types of relational and non-relational databases. JDBC (relational), MongoDB (NoSQL), Cassandra/Thrift (NoSQL), Cassandra/Hector (NoSQL), and Hbase (NoSQL) API's have been implemented. Table 3.2.1.3.1 demonstrates the key APIs that are available for rest of MDaaS.

Table 3.2.1.3.1: DAL APIs

Method	Description	Return Value	Argument List		
			Type	Name	Purpose
initialize	Method to initialize a DAL connection. It can be used to create, read or write from a	Object. Different types of implementations of DAL, viz. Jdbc,	IN	schemaName	Name of schema (analogous to keyspace in NoSQL)
			IN	tableName	Name of table (analogous to Column Family in

	database. Upon success, a connection handle is returned. Applications are expected to use this handle for subsequent calls to the database.	Hector, Thrift, Hbase, etc have their own ways to provide database connection. A generic type, Object is returned, so that upper layers of applications do not need to understand the underlying implementation.	IN	tableName	NoSQL)
			IN	createSchema	Flag to indicate whether a new schema (or keyspace) needs to be created
			IN	createTable	Flag to indicate whether the table specified needs to be created.
			IN	isAscii	Flag when set indicates that the new table being created is of ascii type. If set to 'false' creates a table with CounterColumnType.
			IN	verbose	Flag to enable additional printing of diagnostic messages
	Prototype: `abstract Object initialize(String schemaName, String tableName, boolean createSchema, boolean createTable, boolean isAscii, boolean verbose);`				
destroy	Destroys the specified connection from a database. After this is called, no further calls to the database can be made using the handle.	None	IN	handle	The handle that was returned by an earlier call to *Initialize*.
	Prototype: `abstract void destroy(Object handle);`				
addColumn	Adds a specified column to a specified table.	True if successful, false otherwise	IN	handle	Database connection handle as returned by 'initialize'
			IN	tableName	Name of table (column

					family) where new column is to be added
			IN	colName	Name of column to be added
			IN	verbose	Flag when set prints more diagnostic messages about the operation
	Prototype: `abstract boolean addColumn(Object handle, String tableName, String colName, Boolean verbose);`				
insert	Inserts one or more rows into a Database	True if successful, false on failure	IN	handle	Database connection handle as returned by 'initialize'
			IN	tableName	Name of table (column family) where data is to be inserted
			IN	row	For NoSQL, this accepts a HashMap in the <key,<name,value>> format. There can be multiple instances of <name,value> in the list for each key. Insertion of multiple keys is allowed. For RDB (Jdbc) insertion is done through a callback, refer JdbcCallbacks.
			IN	verbose	Flag when set prints more diagnostic messages about the operation
	Prototype: `abstract void insert(Object handle, String tableName, HashMap<String, HashMap<String, String>> row, boolean verbose);`				
delete	Deletes rows matching a specified	True if successful, false	IN	handle	Database connection handle as returned by 'initialize'

delete	primary key from a database	otherwise.	IN	handle	Database connection handle as returned by 'initialize'
			IN	tableName	Name of table (column family) where from rows are to be deleted.
			IN	pKey	Primary key matching which rows are to be deleted.
			IN	verbose	Flag when set prints more diagnostic messages about the operation
	Prototype: `abstract void delete(Object handle, String table, String pkey, boolean verbose);`				
query	Queries a specified database.	Set of rows matching the given condition.	IN	handle	Database connection handle as returned by 'initialize'
			IN	tableName	Name of table (column family) where from rows are to be queried.
			IN	pKey	Primary key matching which rows are to be deleted. If set to NULL, all rows from the specified table are returned.
			IN	verbose	Flag when set prints more diagnostic messages about the operation
	Prototype: `abstract HashMap<String, HashMap<String,String>> query(Object handle, String table, String pKey);`				

3.2.2 Mobile Data Infrastructure Cloud – MDIC

This chapter enumerates the detailed functional design of the Mobile Data Infrastructure Cloud (acronym MDIC), which forms the backbone of the MDaaS solution. This chapter has been split into a number of logical sections based on the functionality of different components, such as how MDIC connects to the BDC, its query processing semantics, how transactions are handled, how multiple tenants are managed, etc. The design presented here will be used by the mobile clients (design of which is described in Chapter 3.2.3) to communicate with MDaaS services.

3.2.2.1 BDC Connectivity Agent

The Backend Database Cloud Connectivity Agent is responsible for communications between the MDIC and BDC. A BDCIORequest determines the BDC target server and sends the I/O to that server through the BackendDataCloudAbstractionLayer. This layer enables MDIC to interface with any cloud without requiring specific code for the backend cloud in the remaining portions of MDIC. Figure 3.2.2.1.1 shows the design of this component.

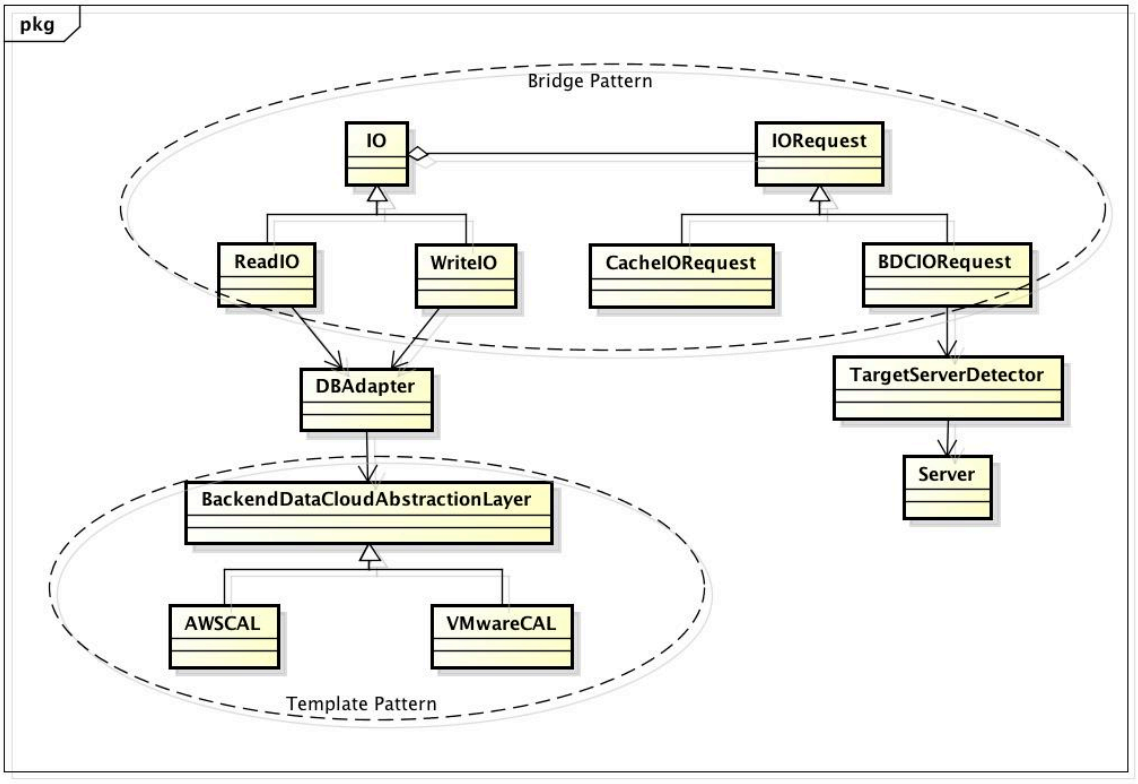


Figure 3.2.2.1.1: BDC Connectivity Agent

3.2.2.2 RMDIC/CMDIC Connectivity Agent

The connectivity between the different types of MDICs is a crucial aspect of the design. Figure 3.2.2.2.1 illustrates the high-level connectivity model between the mobile unit, CMDIC and RMDIC. The steps shown in the figure are enumerated as follows:

- The MDaaS aware applications need to make an initial call to the MDaaSClient class to initialize the MDaaS calling handle as follows:
  - `MDaaSClient mdaasClient = new MDaaSClient();`
- The MDaaSClient constructor will have the hardcoded IP address of CMDIC, and internally calls the web service endpoint getRMDICInstance.
- The getRMDICInstance() endpoint will call ClientRequest class in the above figure. CMDIC will find out the nearest RMDIC and will return its IP Address.
- The MDaaSClient constructor will save the IP address in the object returned to the client.
- Subsequent calls to MDaaS can be made as follows
  - `handle = mdaasClient.initialize(param1, param2);`
  - `mdaasClient.queryDB(handle, …);`

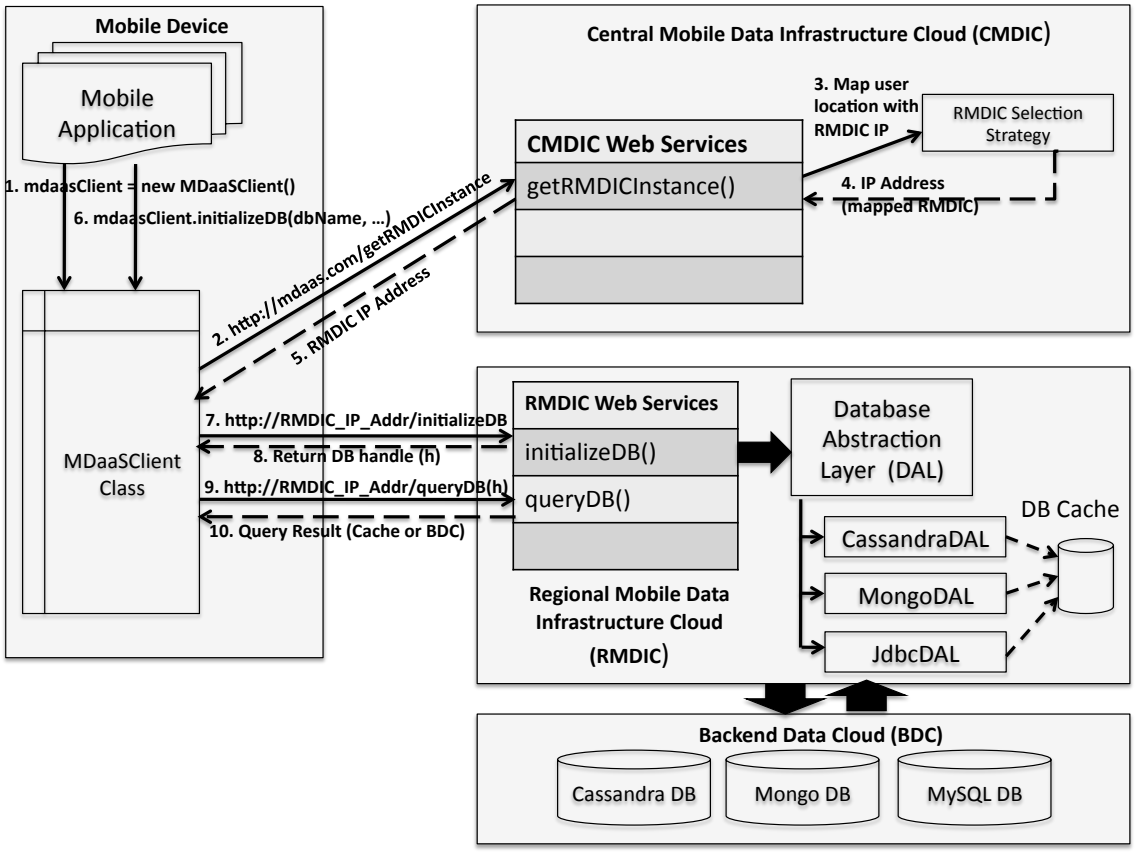


Figure 3.2.2.2.1: Mobile Client – CMDIC – RMDIC API Access Model

Some considerations for management of the connection and handle are as discussed below:

- **Liveliness of Connections** – the use of database handles enables clients to NOT having to maintain connections with the MDaaS server at all times as the server will use the handle and map it with the respective mobile client during each call.
- **Life-cycle management for the handle at the RMDIC**:
    - *How does the server (RMDIC) know that the client has gone away*?
        - Client has sent an explicit logout
        - Session ID expiry – after some time of inactivity (may be made configurable in future)
        - When the client has moved from one RMDIC to another (see below).
    - *How does the client know that it has to use a new handle?* This will be in two situations, when:
        - Server should return an error code during queries if the client has been inactive for a long time. At that point the client should request a new handle.
        - The client is moved from a RMDIC to another (refer Alternative 2 below).
    - *How does the server know whether the client has moved from jurisdiction of one RMDIC to another?*
        - Alternative 1: When turn-around time of responses for responses from RMDIC exceeds a particular value, the client will send a request to CMDIC to getRMDICInstance.
        - Alternative 2: The current RMDIC should make a recommendation based on the client's location. This can be achieved by having the client send the local geo location along with every request the client makes to the RMDIC. If the RMDIC detects that the client is outside its jurisdiction, it sends a request to CMDIC to move the client to the appropriate RMDIC.

We identify below the different use cases of MDIC connectivity and provide a class diagram that illustrates how this is designed (refer to Figure 3.2.2.2.2):

- *User mobility*: In steady state the system must select a group of RMDICs around a user's location, so that when the user moves around, the user can interact with the new RMDIC for improving performance. In order to materialize this, RMDIC must maintain its list of peers pertaining to its own user location. This is implemented by RMDICPeerSelector. Any data exchanged by the mobile unit and MDIC should be synchronized, which is implemented by RMDICSynchronizer.
- *Resiliency:* The CMDIC must maintain a health check of all RMDICs in its domain so that in the event of failure of a RMDIC, requests can be handled by a neighboring RMDIC.
- *Servicing user requests:* CMDICs accept user request and forwards it to the appropriate RMDIC. The target RMDIC processes the request, and directly responds to the client (Direct Server Return).

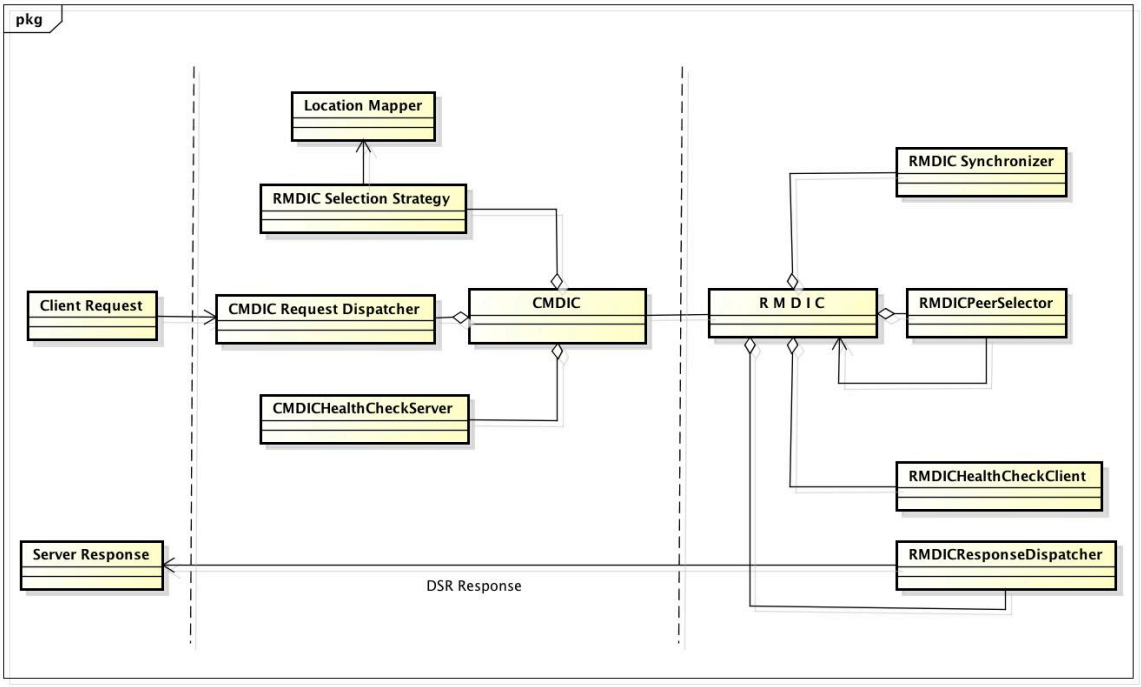


Figure 3.2.2.2.2: CMDIC – RMDIC – Mobile Unit Connectivity Class Diagram

3.2.2.3 Data Filter Agent

The Data Filtering Agent is responsible for creating a location based caching system in MDaaS. With location based caching, MDaaS will predict the content that will be populated in each of the locations where the mobile user is expected to visit during a period of time [5]. This can realized through a *Strategy Design Pattern* as shown in Figure 3.2.2.3.1 which has the ability to select what data gets populated in the cache depending on user's *Home* Profile or *Away* Profile.

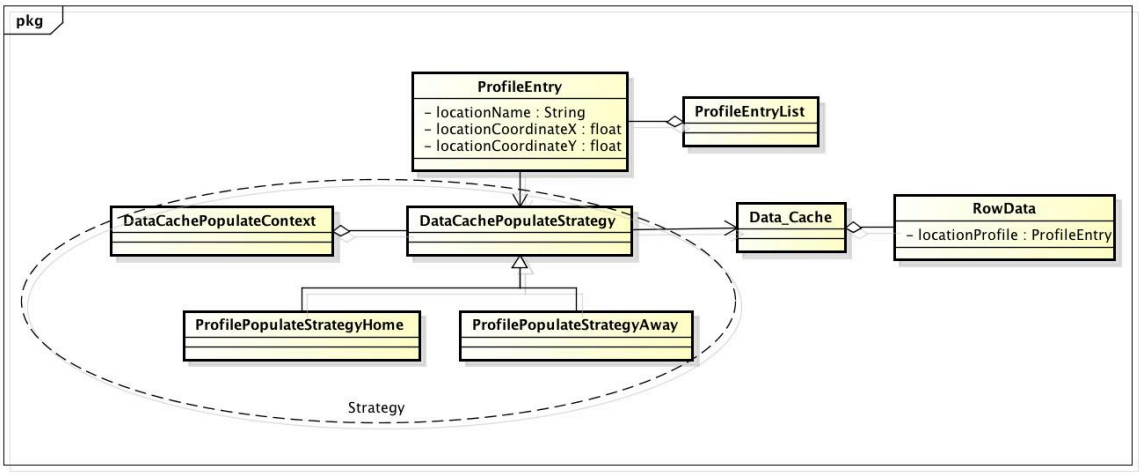


Figure 3.2.2.3.1: Data Filtering Agent

3.2.2.4 Data Caching Agent

The cache management technique in MDaaS layer is based on *Hot Data Caching* mechanism as documented in [43]. In this model, data that is requested by *most* of the transactions from a mobile device is labeled as a *hot data* and is cached depending on the physical size of the cache at MDaaS. These transactions are called *active transactions* and use the hot data. They will be executed first by MDaaS so as to build a data pre-fetching stream for the remaining transactions, which are placed in a *waiting queue.* Once transactions from the *active transaction queue* finishes, more transactions are scheduled in for execution. The MDaaSDataCache class is a listener for the BackendDatabaseTable class for each RowID that identifies each row of each table.

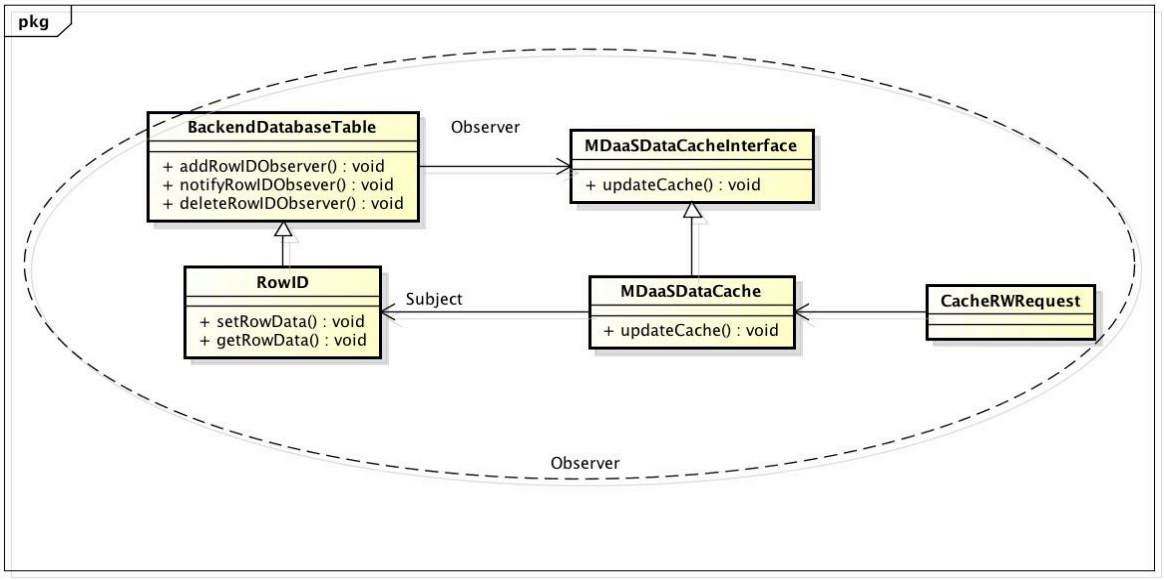


Table 3.2.2.4.1: Cache Management Agent

3.2.2.5 Data Shard Agent

Regional Mobile Data Infrastructure Cloud (RMDIC) will be comprised of a number of constituent nodes that will have the capability to shard *cached* data in the event of data growth. The shards will be balanced periodically depending on the sharding strategy. Conceptually we can think that data can be sharded statically or automatically beyond a threshold as discussed in [44]. A RMDIC wide data cache shard list will be maintained as shown in Figure 3.2.2.5.1.

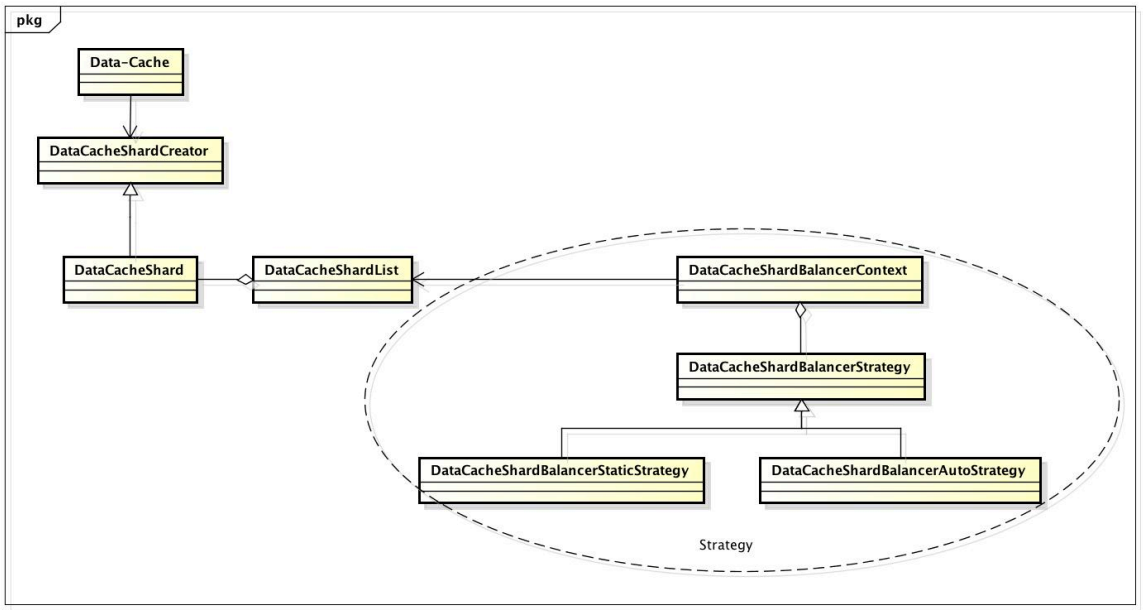


Figure 3.2.2.5.1: Data Sharding Agent

3.2.2.6 Data Resiliency Management Agent

The ability of the MDaaS product to provide a mechanism for retrieving data from a failed RMDIC node is possible owing to the Data Resiliency Management Agent. In the case of failure of a *Regional MDIC,* all client requests should be automatically forwarded to the nearest available RMDIC so that client requests are serviced seamlessly. As discussed in [50], the *Backup RMDIC* replicates the data from the *Active RMDIC* so that in the event of a failure, it can take up the role of an active RMDIC. RMDICSelectionStrategy runs in the *Central MDIC (CMDIC)* and it implements *Strategy Design Pattern* as shown in Figure 3.2.2.6.1. Once the *target* RMDIC is identified, the request is dispatched to that RMDIC, which ultimately provides the response.

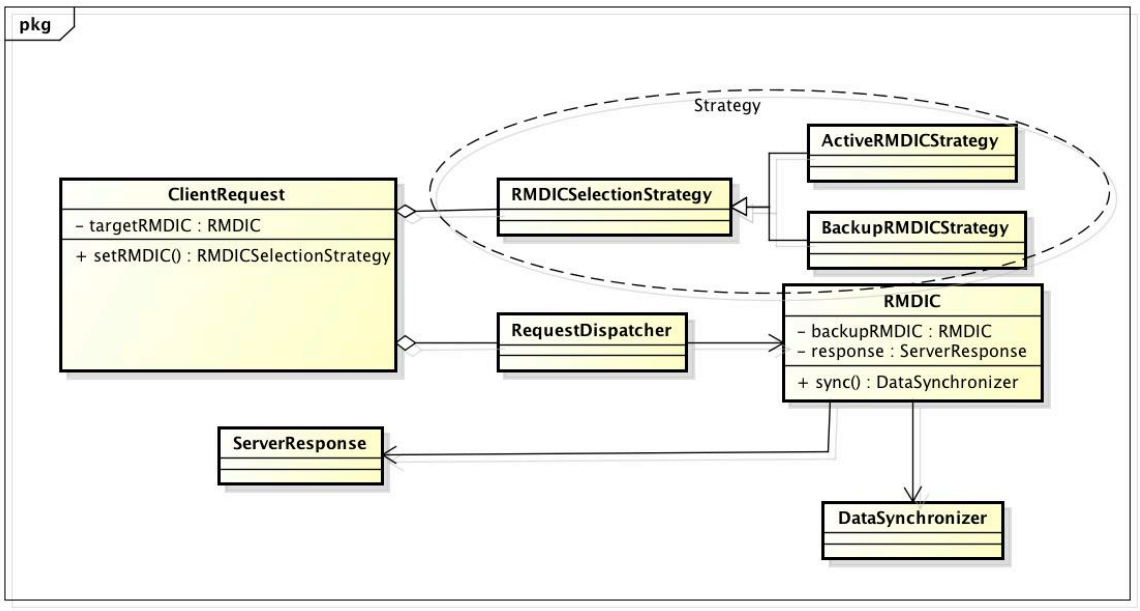


Figure 3.2.2.6.1: Data Resiliency Management Agent

3.2.2.7 Data Search Acceleration Agent

Indexing is a popular model for improving performance of searches. There are two forms of indexing [4]:

- *Air Indexing:* In this form of indexing, the mobile client sends a query to the server, which pre-computes index information and interleaves it with its initial response. This index advises the client when the data will arrive, and mobile clients can save battery power by not listening on the channel all the while for data to arrive. Many mobile carriers while implementing LDIS use this mechanism. The drawback of this approach is that the additional indexing information can make the initial responses longer.
- *Disk Indexing:* Most mobile databases add an index attribute to the data and store the index along with the data so as to make retrieval of data faster.

Air level indexing causes access latency and allows only sequential access. Thus it is not used in practical implementations. Disk indexing is more popular and the architecture of hierarchical storage scheme shown in Figure 3.2.2.7.1 can be extended at the server side to depict how Disk Indexing is done for RMDIC server S2 that serves three cells (P1, P2, and P3) and RMDIC server S3 that serves another three cells (P4, P5 and P6).

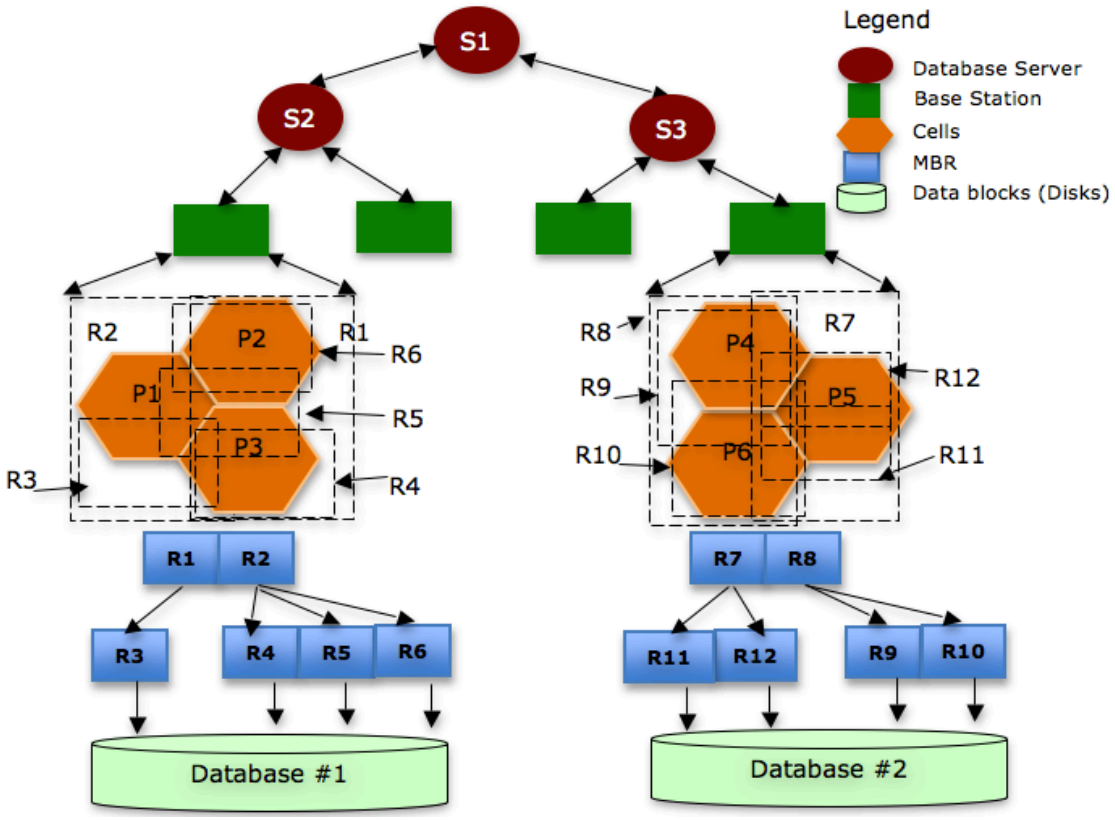


Figure 3.2.2.7.1: Disk Indexing for improving performance

3.2.2.8 Multi-tenancy Agent

Multi-tenancy [1] is a mechanism by which a customer's data is collocated with others without compromising privacy and security. In this section, we will examine various design considerations for handling multi-tenancy of data, its data model and query management in MDaaS solution. The following are some of the key design considerations in multi-tenancy management:

- *Ability to provide optimal performance*: As multi-tenancy will involve lookup of metadata tables, real-time performance of data access may be impacted. A key design consideration is the ability to provide high performance with different forms of multi-tenancy.
- *Ability to scale*: As mobile traffic pattern can be spiky, and often bursty, there should be means to manage multi-tenancy by scaling out versus scaling up services in MDICs.
- *Rules and thresholds for spliting or sharding table*: For performance reasons, customers in a geographic region will have their data near to them, but in mobile environment there is possibility that users move between geographic regions. A consideration arises when tenants move outside of home area.
- *Ability to extend table size and add custom columns:* For flexibility reasons, customers may want to custom their own tables. If their data is collocated with others, the design should be able to accommodate this without affecting data of other tenants.
- *Backup and recovery*: Backup and recovery of data may be directly tied to the billing and subscription policy of a customer. With multi-tenant managed data, it is a challenge to be able to extract customer-specific data from a shared table and then perform backup only for that data.
- *Locality of multi-tenancy lookups*: As multi-tenancy involves additional metadata lookups, a key design consideration that can directly translate to performance is where the looks are done, whether data is duplicated on the RMDIC datacenters or remotely at the BDC?

3.2.2.8.1 Types of Multi-tenancy

We define three types of multi-tenancy data model as follows:

- Type-1: Private-Schema Multi-tenancy
- Type-2: Shared-Schema Multi-tenancy
- Type-3: Shared-Table Multi-tenancy

3.2.2.8.1.1 Type-1: Private-Schema Multi-tenancy Model

In *Private-Schema Multi-tenancy* model, the same database is used but data is stored on different schema, ie, each tenant owns his or her own schema space. This allows complete isolation of data for a customer. MDaaS users view data belonging to this as *Private Schema, Private Table* category. An illustration of schema-based multi-tenancy model is shown in Figure 3.2.2.8.1.

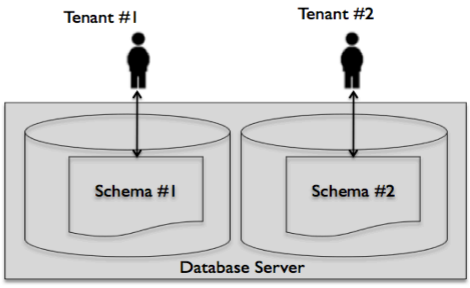


Figure 3.2.2.8.1: Type-1: Private-Schema Multi-tenancy Data Model

3.2.2.8.1.2 Type-2: Shared-Schema Multi-tenancy Model

In *Shared-Schema Multi-tenancy* model, multiple tenants share the same schema but are always located in their respective private tables. It is possible that a tenant's table is optimized with some of their other tables, but they never share a table with another tenant. A Tenant ID column is maintained as a tag on every table [61] [62] [63] to differentiate between different tenants. An illustration of this model is shown in Figure 3.2.2.8.2.

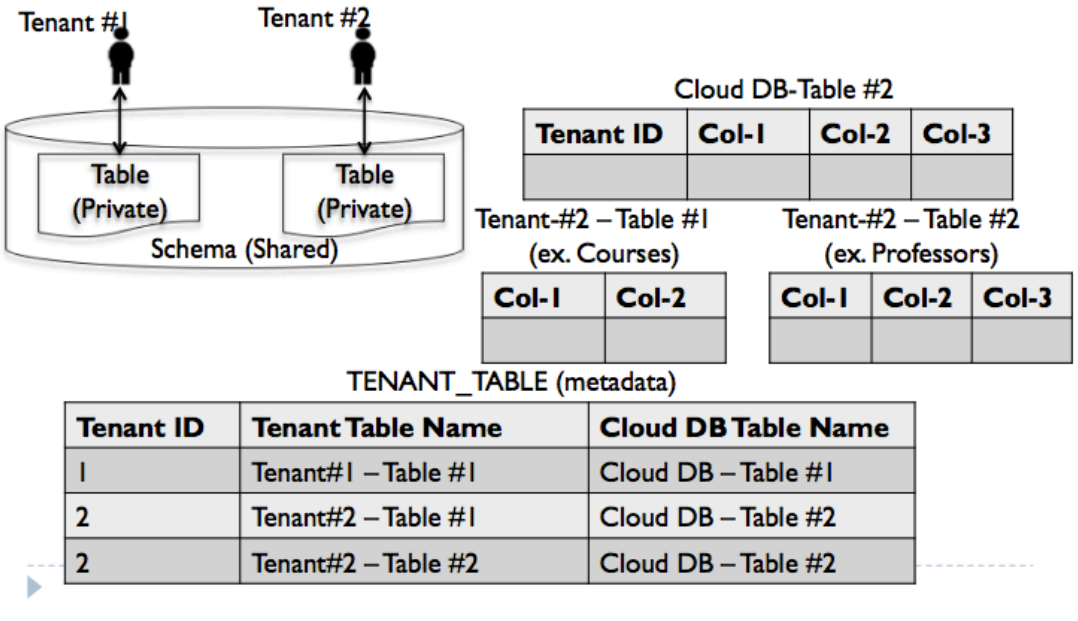


Figure 3.2.2.8.2: Type-2: Shared-Schema Multi-tenancy Data Model

3.2.2.8.1.3 Type-3 Shared-Table Multi-tenancy Model

In *Shared-Table Multi-tenancy Model,* a tenant can share tables with other tenants. Each row of a table has tenant specific metadata that helps identify which tenant that row belongs to. This model of multi-tenancy optimizes space and improves efficiency of database queries for MDaaS. A sample schematic representation of meta-data management is shown in Figure 3.2.2.8.3.

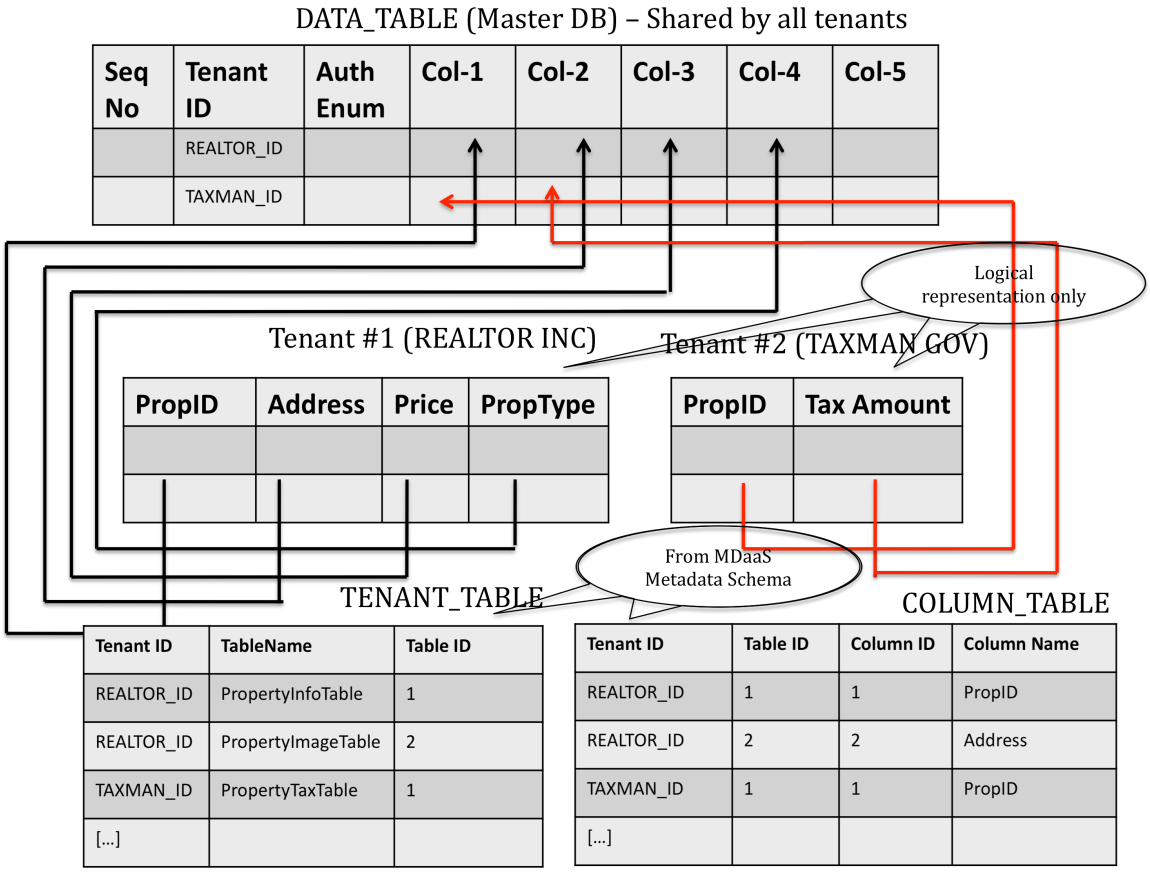


Figure 3.2.2.8.3: Type-3: Shared-Table Multi-tenancy Data Model

Type	Name	Description	DB Server Shared?	Schema Shared?	Table Shared?	TCO
Type-1	Private-Schema	Customer gets a private (dedicated) schema	Yes	No	No	High
Type-2	Shared-Schema	Customer shares the schema with other tenants, but have private tables	Yes	Yes	No	Medium
Type-3	Shared-Table	Customer shares the table with other tenants	Yes	Yes	Yes	Low

3.2.2.8.2 Data Model

The following metadata tables are maintained:

- *Tenant Table:* In Type-2 Multi-tenancy, this table is used to map table names with respective tenants. Similarly in Type-3 Multi-tenancy, this table is used to create a mapping of the database names, the tables and their tenant IDs. For example, PropertyInfoTable is the table created by the DBA of Realtor Inc, while PropertyTaxTable is the table created by the DBA of Santa-Clara-County-Property-Tax (a government organization), who shares the same schema and table with Realtor Inc.
- *Column Table:* This table maintains a mapping of the column names with their column IDs.
- *Data Table:* This table contains the actual data.

A Jason representation of the MDaaS metadata tables for managing multi-tenancy data models is shown in Figure 3.2.2.8.4.

```
TENANT_TABLE
    { "tenantId" : ”REALTOR-INC", "password" : "1234", ”dbName” : "RealEstateDB",
      "tableName" : "PropertyInfoTable", "tableID" : “tbl1"}
    { "tenantId" : ”SANTA-CLARA-COUNTY-PROPERTY-TAX", "password" : "1234”,
      ”dbName" : "RealEstateDB”, "tableName" : ”PropertyTaxTable", "tableID" : “tbl2”}
COLUMN_TABLE
    { "tenantId": "t1", "columnId": "c1", "columnName" : ”prop_id”, "columnDataType" : ”string" }
    { "tenantId": "t1", "columnId": "c2", "columnName" : ”price”, "columnDataType" : ”double" }
    { "tenantId": "t1", "columnId": ”c3", "columnName" : "address", "columnDataType" : "string" }
    { "tenantId": “t2", "columnId": ”c1", "columnName" : ”prop_id”, "columnDataType" : "string" }
    { "tenantId": “t2", "columnId": ”c2", "columnName" : ”property_tax", "columnDataType" :
    ”double" }
        [..]
DATA_TABLE
    { “tenantId”: “REALTOR-INC”, “tableId”: “tbl1”, “recordId” : “1”, “columnId”: “c1”,
    “columnValue”: ”PROP_1” }
    { “tenantId”: “REALTOR-INC”, “tableId”: “tbl1”, “recordId” : “1”, “columnId”: “c2”,
    “columnValue”: ”640000” }
    { “tenantId”: “REALTOR-INC”, “tableId”: “tbl1”, “recordId” : “1”, “columnId”: “c3”,
    “columnValue”: ”10100 Danube Drive, Cupertino, CA 95014” }
    { “tenantId”: “SANTA-CLARA-COUNTY-PROPERTY-TAX”, “tableId”: “tbl2”, “recordId” : “1”,
    “columnId”: ”c1”, columnValue”: ”PROP_1” }
    { “tenantId”: “SANTA-CLARA-COUNTY-PROPERTY-TAX”, “tableId”: “tbl2”, “recordId” : “1”,
    “columnId”: ”c2”, columnValue”: “3800” }
    { “tenantId”: “SANTA-CLARA-COUNTY-PROPERTY-TAX”, “tableId”: “tbl2”, “recordId” : “2”,
    “columnId”: ”c1”, columnValue”: ”PROP_2” }
    { “tenantId”: “SANTA-CLARA-COUNTY-PROPERTY-TAX”, “tableId”: “tbl2”, “recordId” : “2”,
    “columnId”: ”c2”, columnValue”: “4000” }
        [..]
```

Figure 3.2.2.8.4: JASON representation of metadata for multi-tenancy

MDaaS will maintain a global metadata table called TENANT_TABLE wherein it will store information about all its tenants. This will be used to vector into the type of multi-tenancy schema (viz. one of Type-1, Type-2 or Type-3) while handling a query. TENANT_TABLE is maintained in a separate schema called MDaaS-Metadata-Schema by MDIC. During schema creation, tenants can specify which tables will reside with what level of multi-tenancy. For Type-1 Multi-tenancy, tenants are provided with their own private schema. For Type-2 Multi-tenancy, tenant tables are collocated within the same schema, but tables are tagged with Tenant_ID prefix so that lookups are faster. Finally,

for Type-3 Multi-tenancy, a COLUMN_TABLE (metadata) is maintained by MDaaS, which holds which column mapping and actual data is stored in DATA_TABLE. The data model design of the three types of multi-tenancy is shown in Figure 3.2.2.8.5.

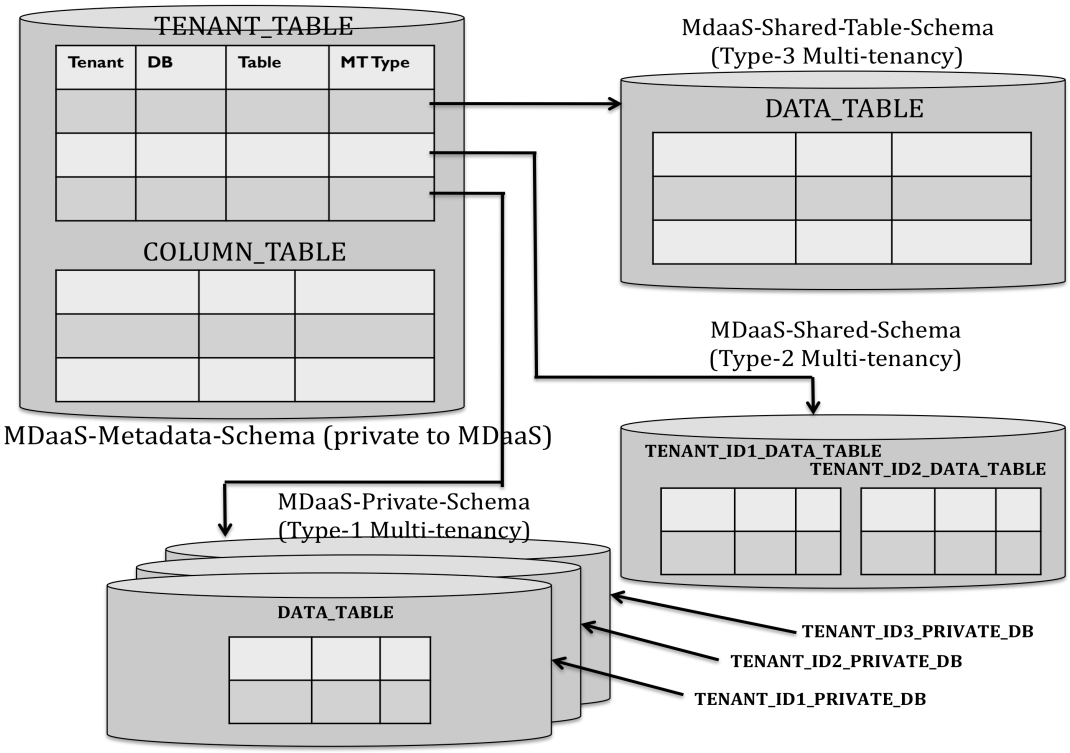


Figure 3.2.2.8.5: Data Model for MDaaS Multi-tenancy

3.2.2.8.3 New Tenant Creation

New tenant creation involves adding an entry for the tenant in the TENANT_TABLE of MDaaS-Metadata-Schema as shown in Figure 3.2.2.8.5. When a new tenant is created, the administrator must specify the Database name(s) requested by the tenant and an entry is added to the metadata table. Later a tenant can add their tables and specify multi-tenancy types for those tables. At that time the TENANT_TABLE entries are modified.

3.2.2.8.4 Schema Creation

Multi-tenant schema can be created by two mechanisms:

- *Through Admin UI:* This method is useful while adding smaller number of entities (schema columns or tables). In the UI based method, schema is created through the MDaaSAdmin app that creates entries for the new schema in the TENANT_TABLE as well as in the COLUMN_TABLE. The flow for adding a new tenant schema is shown in Figure 3.2.2.8.6.

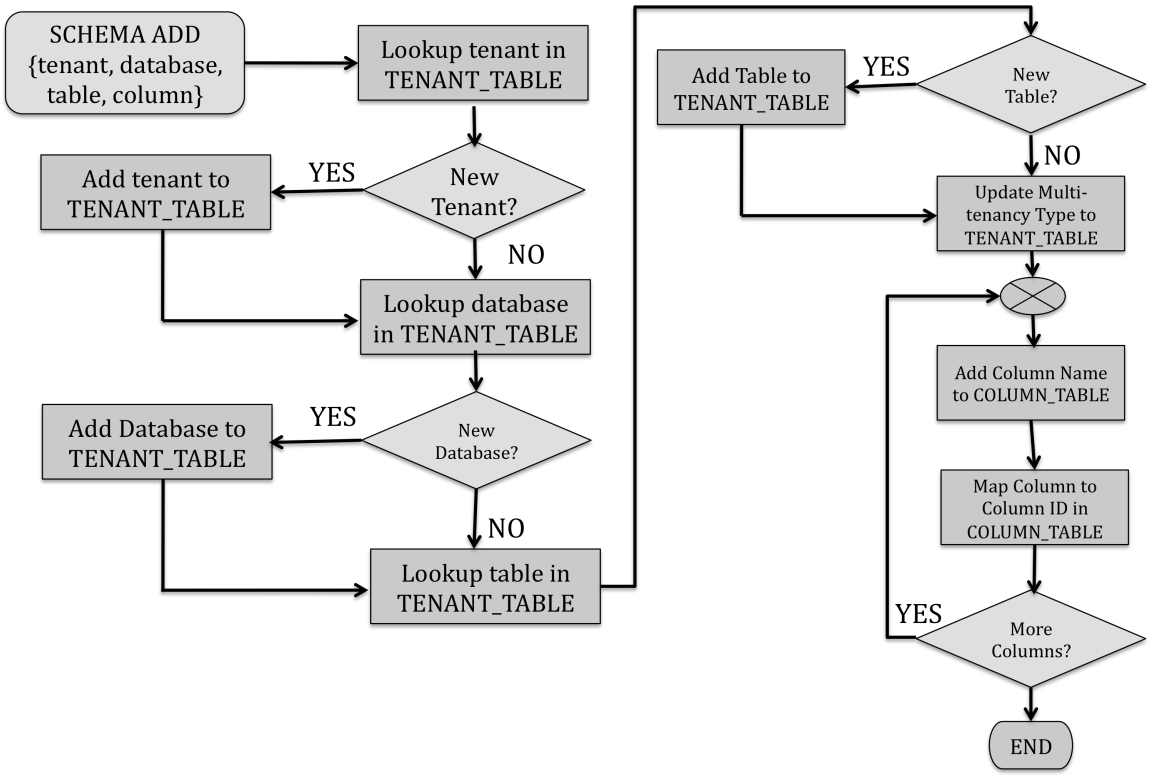


Figure 3.2.2.8.6: New Tenant/Schema Addition Control Flow

- *Through Automated Deployment Script:* This method is useful while creating large number of entities in an automated fashion. This is done through an XML input file can be validated through a configuration script. A sample configuration file and its validation script is illustrated in Figure 3.2.2.8.7 and 3.2.2.8.8.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="mdaasdatabase">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="tenant" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="name" type="xs:NMTOKEN" use="required" fixed
="MDaaS" />
                <xs:attribute name="type" type="nosql" use="required" />
        </xs:complexType>
</xs:element>

        <xs:simpleType name="nosql">
                <xs:restriction base="xs:string">
                <xs:enumeration value="Mongo"/>
                <xs:enumeration value="Cassandra"/>
                </xs:restriction>
        </xs:simpleType>

<xs:element name="tenant">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="tenantid" />
                        <xs:element ref="password" />
                        <xs:element ref="database" />
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="tenantid">
        <xs:complexType mixed="true" />
</xs:element>

<xs:element name="password">
        <xs:complexType mixed="true" />
</xs:element>

<xs:element name="database">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="databasename" />
                        <xs:element ref="table" maxOccurs="unbounded" />
                </xs:sequence>
        </xs:complexType>
```

Figure 3.2.2.8.7: Sample XSD file for Multi-tenancy Schema creation

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<mdaasdatabase name="MDaaS" type="Mongo"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mdaas-db-schema.xsd">

<tenant>
        <tenantid>REALTOR-INC</tenantid>
        <password>123456</password>
        <database>
                <databasename>RealEstateDB</databasename>
                <table>
                        <tablename>PropertyInfoTable</tablename>
                        <column name="property_id" type="string" />
                        <column name="price" type="double" />
                        <column name="address" type="string" />
                </table>
        </database>
</tenant>

<tenant>
        <tenantid>SANTA-CLARA-COUNTY-PROPERTY-TAX</tenantid>
        <password>123456</password>
        <database>
                <databasename>RealEstateDB</databasename>
                <table>
                        <tablename>PropertyTaxTable</tablename>
                        <column name="property_id" type="string" />
                        <column name="property_tax" type="double" />
                </table>
        </database>
</tenant>

</mdaasdatabase>
```

Figure 3.2.2.8.8: Sample XML file for schema creation of two tenants – REALTOR-INC & SANTA-CLARA-COUNTY-PROPERTY-TAX

Algorithm: Multi-tenant Schema Creation (script based deployment)

Input: XSD_SCHEMA_URL: pointer to the to the system defined XSD file for tenant XML database schema

DB_SCHEMA_URL: pointer to the user defined tenant database file

Output: Multi-tenant Schema created

Begin

1. Validate the system database schema (XSD_SCHEMA_URL)
2. Call XML parser to parse the database file (DB_SCHEMA_URL)
3. foreach tenantID in XML database file

   do

   (a) Lookup tenantID in MDaaS-Metadata::TENANT_TABLE and determine the multi-tenancy type
   (b) Choose the schema where tenant's tables will be placed (one of

```
            Private-Schema-DB, Shared-Schema-DB or Shared-Table-DB)
        (c) Read the XML file for the tenant and add the columns into
            MDaaS-Metadata::COLUMN_TABLE
        (d) If tenancy is Type-1 or Type-2, create the schema in the
            database if needed. Note, for certain backend databases, this
            may be delayed till actual data insertion.
    end-do
End
```

3.2.2.8.5 Data Insertion Mechanism

As mentioned in earlier chapters, data is always inserted into the DATA_TABLE of the respective schema. This is accomplished by looking up the tableID for the specified tenantID, databaseName and tableName, then mapping each column to be inserted to the respective columnID, and then finally calling DAL to complete the I/O. This mechanism is described in the Algorithm discussed below.

```
Algorithm: Multi-tenant Data Insertion
Input:      tenantID – ID of the tenant
            tableName – name of the table where data is to be inserted
            columnList – list of {name, value} pair
Output:     Data Inserted into database, status returned to caller
Begin
1.          (a) Lookup the tableID for the specified tenant tableName
            (b) Lookup the multi-tenancy level based on the tenantID
            from MDaaS-Metadata::TENANT_TABLE
2.          if (multitenancy-level == Type-1)
            then
            (a)   schemaName := MDaaS-Private-Schema + "." + tenantID
            else if (multi-tenancy-level == Type-2)
            then
            (a)   schemaName := MDaaS-Shared-Schema
            else if (multi-tenancy-level == Type-3)
            then
            (a)   schemaName := MDaaS-Shared-Table
            endif
            (b)   tableName := DATA_TABLE
3.          For each row to be inserted, do
            (a)   Generate a new Record ID (or GUID)
            (b)   For each <columnName, columnValue> in the given map
                  do
                  (i)   Given a tableID (from 1a) and columnName, find
                        columnID from COLUMN_TABLE
```

```
                columnID := mapColumnName(MDaaS-
                Metadata::COLUMN_TABLE, columnName)
          (ii)  Insert data to the selected databaseName by
                calling DAL.
                DAL.insert(schemaName, tableName, this_column)
        done
    done
End
```

The above algorithm can be illustrated in the control flow shown in Figure 3.2.2.8.9.

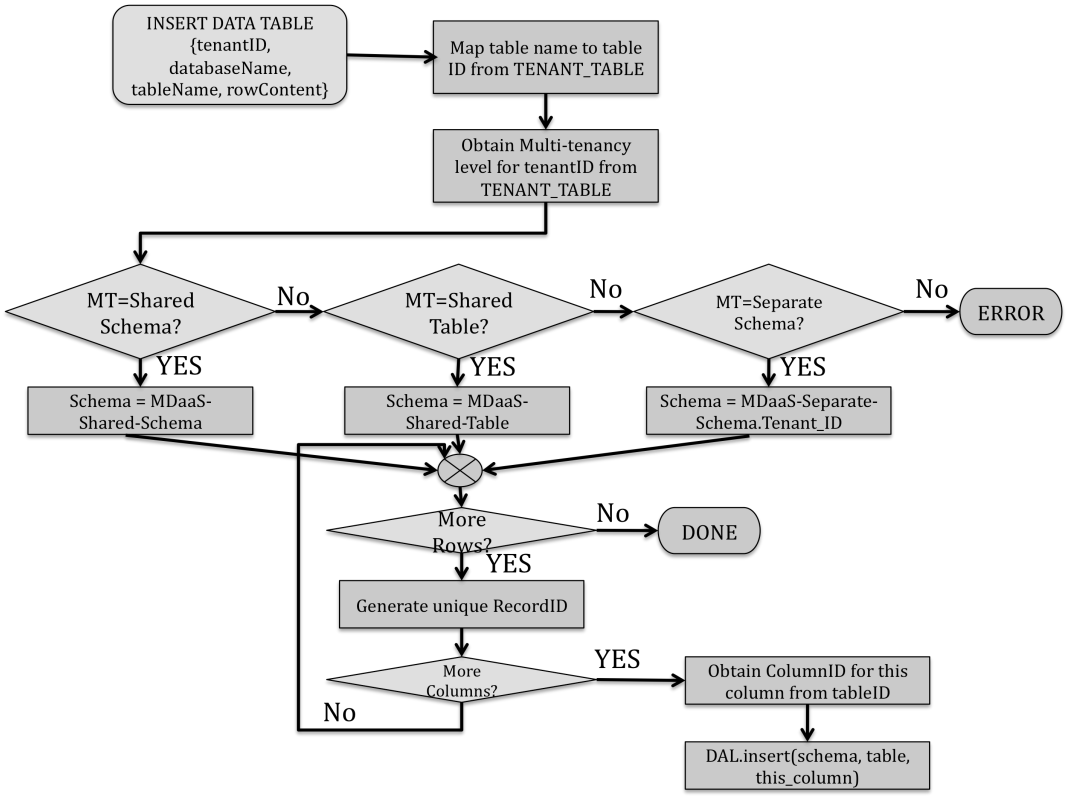


Table 3.2.2.8.9: Data Insertion Control Flow

3.2.2.8.6 Table Customization Mechanism

Table Customization is a mechanism by which a tenant can modify their previously created schema. This can be done through the MDaaS Administration UI. Figure 3.2.2.8.10 illustrates the control flow of this process. As various DATA_TABLEs are designed to contain separate entries for each column, the tables are easily customizable without causing relocation of data within the tables. This makes the design highly scalable.

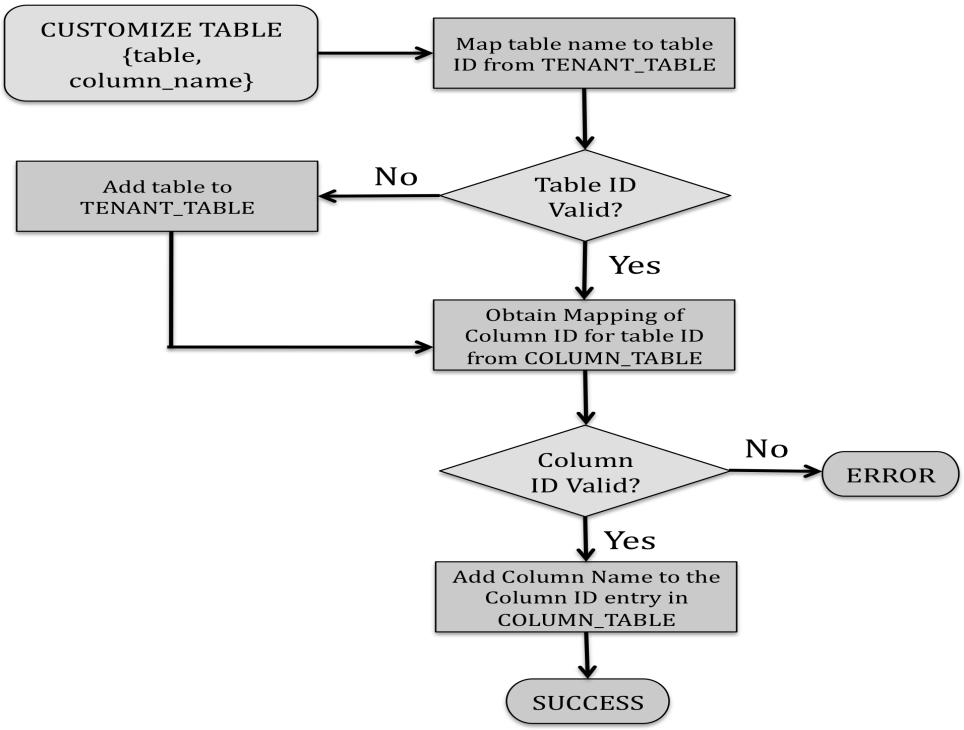


Figure 3.2.2.8.10: Table Customization Control Flow

3.2.2.8.7 Query Mechanism

The query requests are routed from the Web Service Management Agent to the Session Manager, which validates the mobile client and extracts the tenant ID associated with each request. Once the tenant ID is validated and mapped to a specific session, the query is then handed over to Multi-tenancy manager for fulfillment.

Figure 3.2.2.8.11 and Figure 3.2.2.8.12 show the model for initializing and mapping of queries from mobile clients with the multi-tenancy architecture of MDaaS. The steps illustrated in that figure is enumerated as follows:

- During initial connection, MDaaSClient in the mobile device will send its attributes to CMDIC for attachment with getRMDICInstance() call. This information includes the <mac_address, mobile_serial_number, application_id> information.
- During processing of getRMDICInstance, the CMDIC will contact the chosen RMDIC in order to convert the mobile device information into a unique {tenantID, clientID} pair by looking up its private tables, encode it into an unique handle, and send it back to the application along with the response of the getRMDICInstance.
- The application will save the {tenantID, clientID} information in the mdaasClient class.

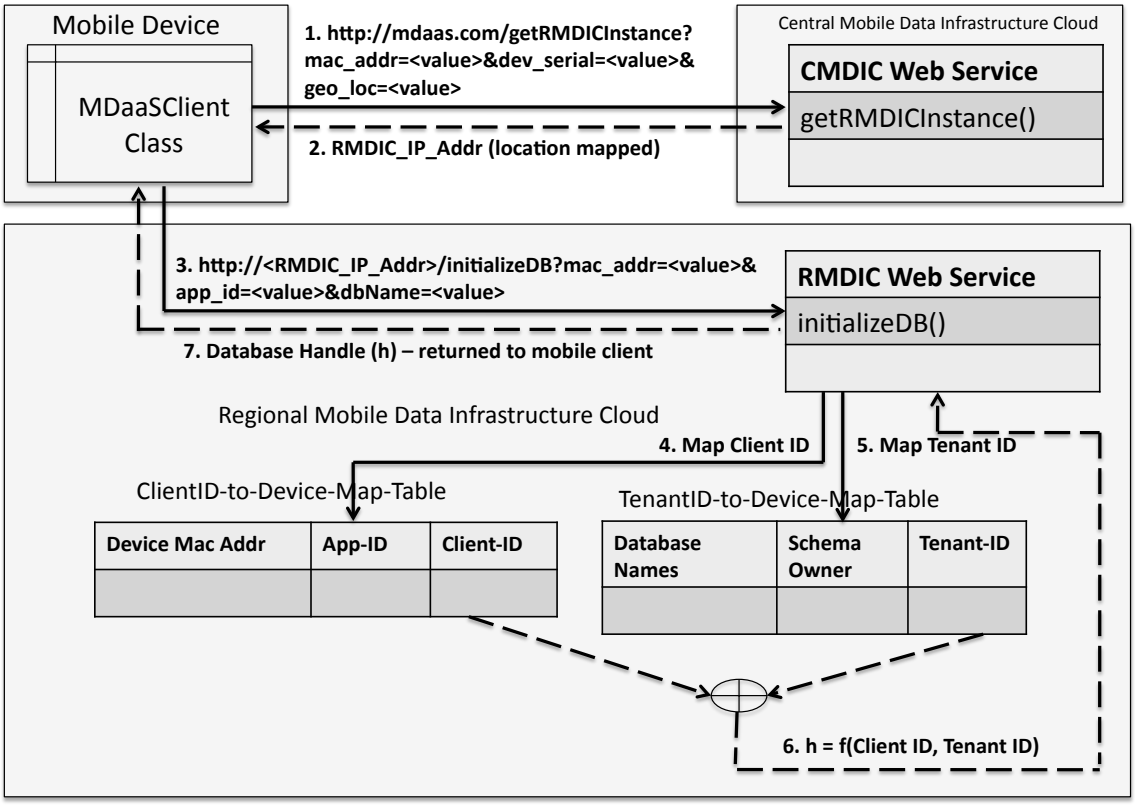


Figure 3.2.2.8.11: Mapping Multi-tenant Data Access Initialization

- During subsequent queries from the application (such as mdaasClient.queryDB()), MDaaSClient object will send the {tenantID, clientID} to the RMDIC.
- As shown in Figure 3.2.2.8.12, RMDIC will use this information to map the query to the right tenant table.
- The model is similar for other database CRUD operations, viz. data insertion, data update and data removal.

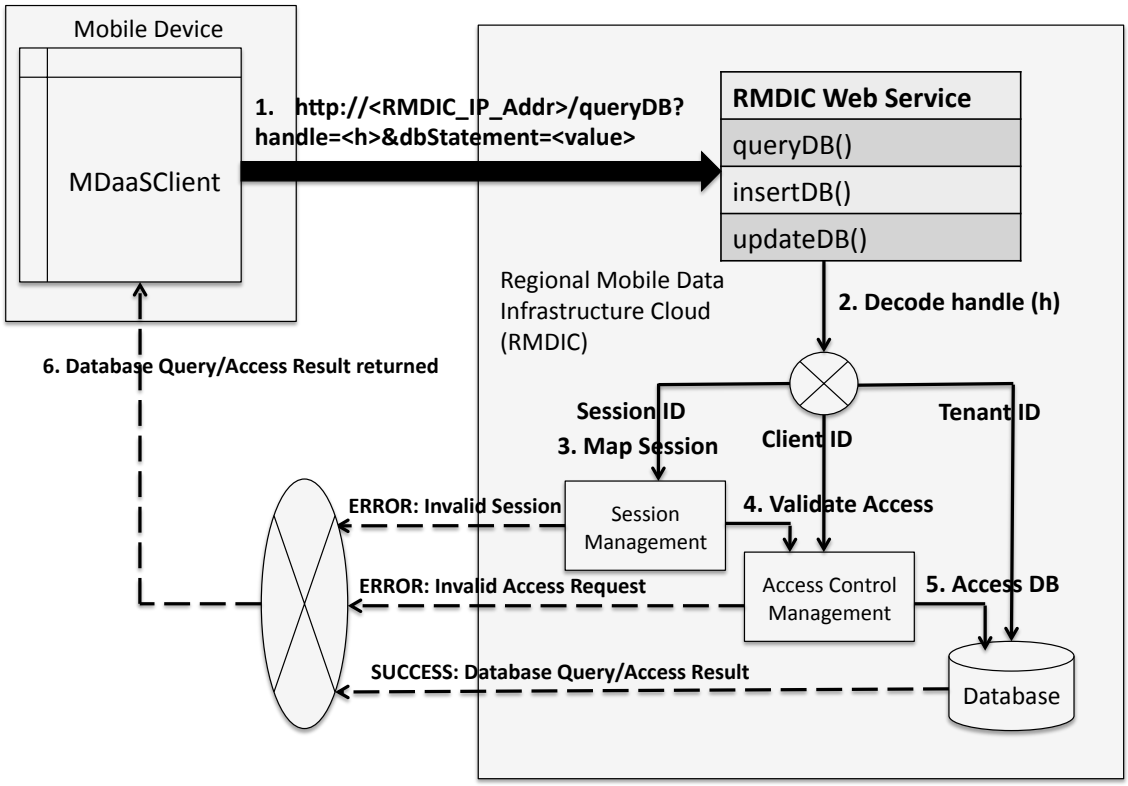


Figure 3.2.2.8.12: Multi-tenant Query Processing Control Flow

```
Algorithm – Multi-tenant Query:
        query(tenantID, dbName, tableName, pKey, startIndex, numRecords)
Input: tenantID – Unique identifier of a tenant
        dbName – Name of the database being queried
        tableName – Name of the tenant's table being queried
        pKey – Primary Key for the query (null if all rows are requested)
        startIndex – Starting index of the row whose data is to bereturned
        numRecords – Number of records to be returned
Output:       Set of records in <key <colName, colValue>[..]> form
Begin
1.      Read the incoming tenantID and look it up in MDaaS-Metadata table. If not
        found, then client is not registered; return failure. Upon success,
        multi-tenancy level (viz. SeparateSchema for Type-1, SharedSchema for
        Type-2, and SharedTable for Type-3) is returned.
2.      if (Multi-tenancy Type == Type-1); then
                (a) Retrieve the database handle for the specified dbName
                (b) Call query of underlying Database Abstraction Layer
                return DAL.query(dbHandle, tableName, pKey, startIndex,
        numRecords)
        else if (Multi-tenancy Type == Type-2); then
                (a) cloudDBTableName = tenantID + "." + tableName
                (b) Retrieve the database handle for the specified dbName
```

```
            (c) Call query of underlying Database Abstraction Layer

            return DAL.query(dbHandle, cloudDBTableName, pkey, startIndex,
            numRecords)

      else if (Multi-tenancy Type == Type-3); then

            (a) Lookup the tableID for the specified tableName of the tenant
                from TENANT_TABLE. Note querying the TENANT_TABLE is a DAL
                request, which may be cached for improving performance.
            (b) Lookup the columnIDs and columnNames for the tableID from the
                COLUMN_TABLE. Note querying the COLUMN_TABLE is a DAL request,
                which may be cached for improving performance.
            (c) foreach column in the list of columnID; do
                        Retrieve all records from DATA_TABLE that matches
                        tableID, columnID and tenantID. This results in a call
                        to DAL.

                        return DAL.query(dbHandleMDaaS-Shared-Table, DATA_TABLE,
                        pKey, startIndex, numRecords)

                end-do

      endif

End
```

3.2.2.8.8 Class Design

Multi-tenancy manager calls the MDaaS *Database Abstraction Layer* to actually retrieve the customer data from the backend database. In effect Multi-tenancy is one layer above the backend storage management and is one step closer to the BDC. Figure 3.2.2.8.13 and 3.2.2.8.14 illustrates the class diagrams of Multi-tenancy management in MDaaS.

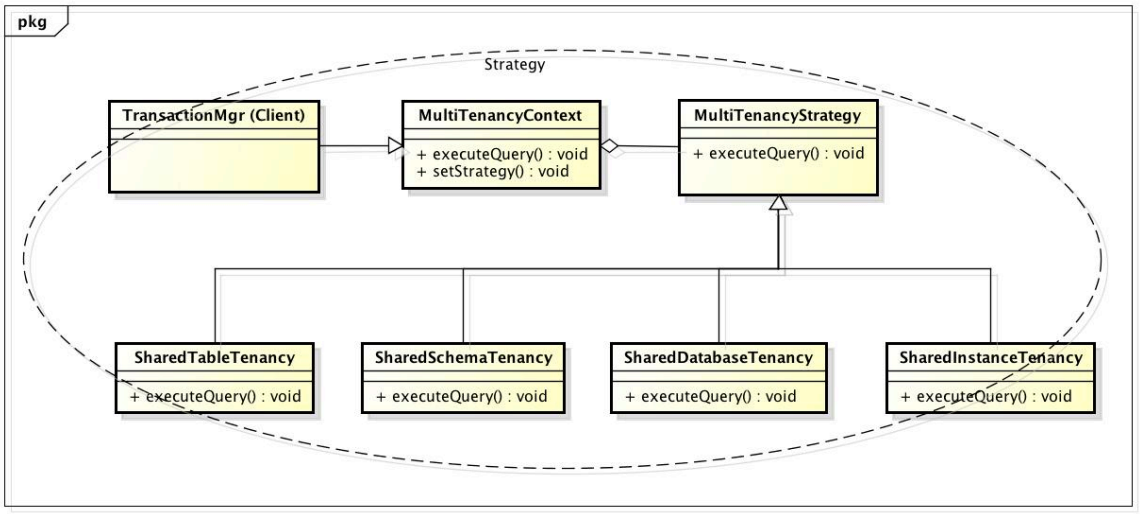


Figure 3.2.2.8.13: Multi-tenancy Query Execution Strategy

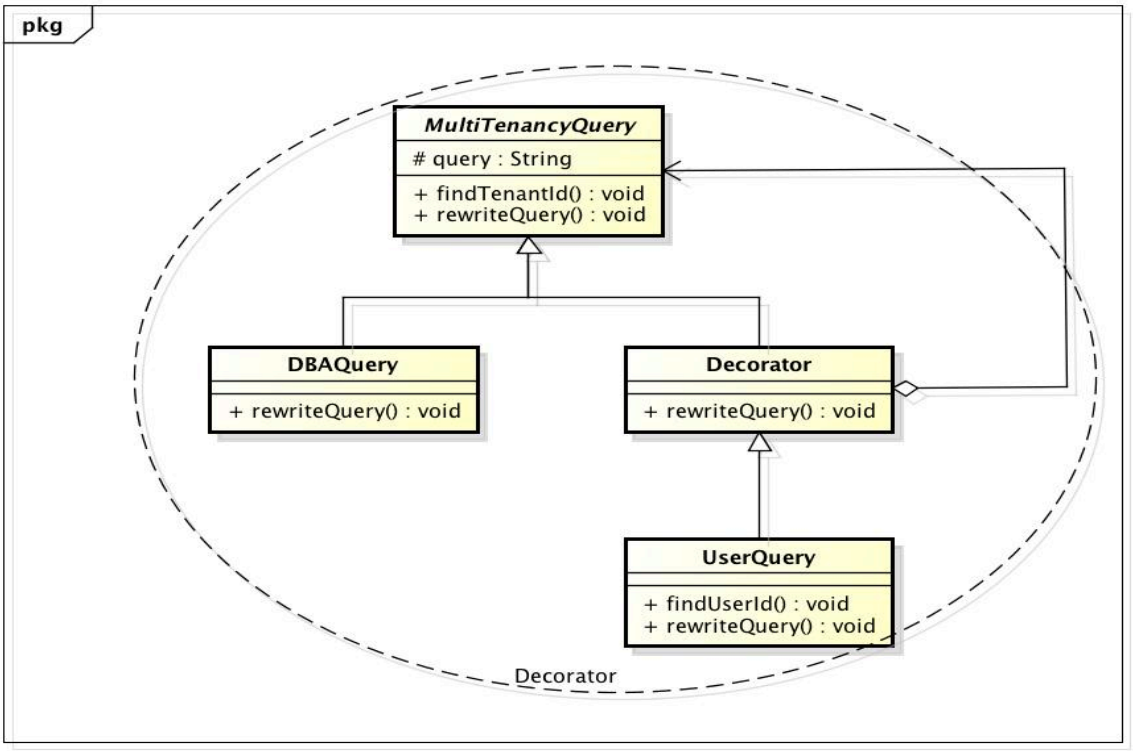


Figure 3.2.2.8.14: Multi-tenancy Query Management

3.2.2.9 Query Management Agent

Mobile database would not know what kind of database is being used at the backend. So MDaaS must have the QM Agent that will translate any user query and send the correct query to the BDC [48]. Mobile device, for example may send a request in MySQL format, which, MDaaS layer must translate into the actual backend database that MDaaS interacts with (which could be Oracle DB). Similarly southbound traffic, viz responses from the BDC towards the mobile units must be translated in the MDIC layer before it is sent toward the actual device. The Query Management Agent can be implemented as shown in Figure 3.2.2.9.1. The QueryGenerator is a *Strategy Design Pattern* that generates the query depending on the type of access. Once generated, the Query is processed by QueryProcessor class, which actually executes the query. The result of the query execution is the DataAccessContext (discussed in Section 3.2.2.11), which actually accesses the data depending on the location.

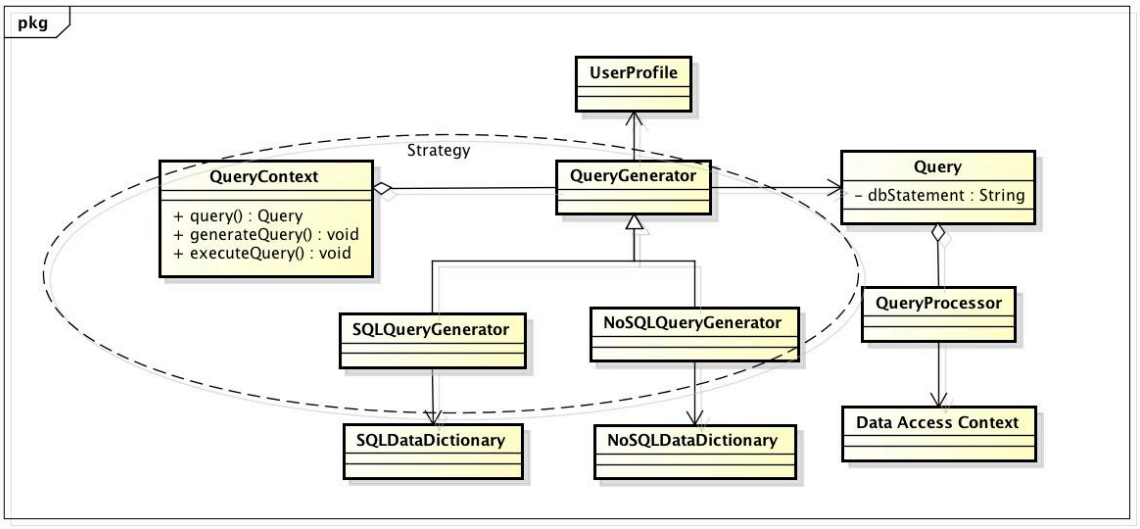


Figure 3.2.2.9.1: Query Management Agent

The translation of mobile database access requests to the format of BDC database will be implemented through *Database Abstraction Layer (DAL)*. DAL will expose database access APIs for southbound (mobile) applications, and have support for Cassandra (Thrift/Hector) database for northbound (BDC) access. Support will be added for the following APIs: initializeDB, createSchema, querySchema, updateSchema, destroySchema, insertDB, deleteDB, queryDB, updateDB. In Figure 3.2.2.9.1, DAL is represented by the QueryGenerator class. The QueryContext class is equivalent to allocDal() which will be used during initializeDB call. Location based queries will be supported by default. For example, if doing 'nearby' query, current geo-locaation must be sent, otherwise target geo-location would be sent by MDaaSClient class.

3.2.2.10 Transaction Management Agent

Transaction Management Agent is responsible for processing transactions in the MDIC layer. Each transaction descriptor contains a TransactionState that can be either *committing* or *receiving*. A batch of transactions contains a number of constituent transactions. As the transactions of a batch continue to arrive, they are saved in the batch and finally committed. The *TransactionList* contains the *in-core* copy of the Transaction Database. The associated classes are shown in Figure 3.2.2.10.1

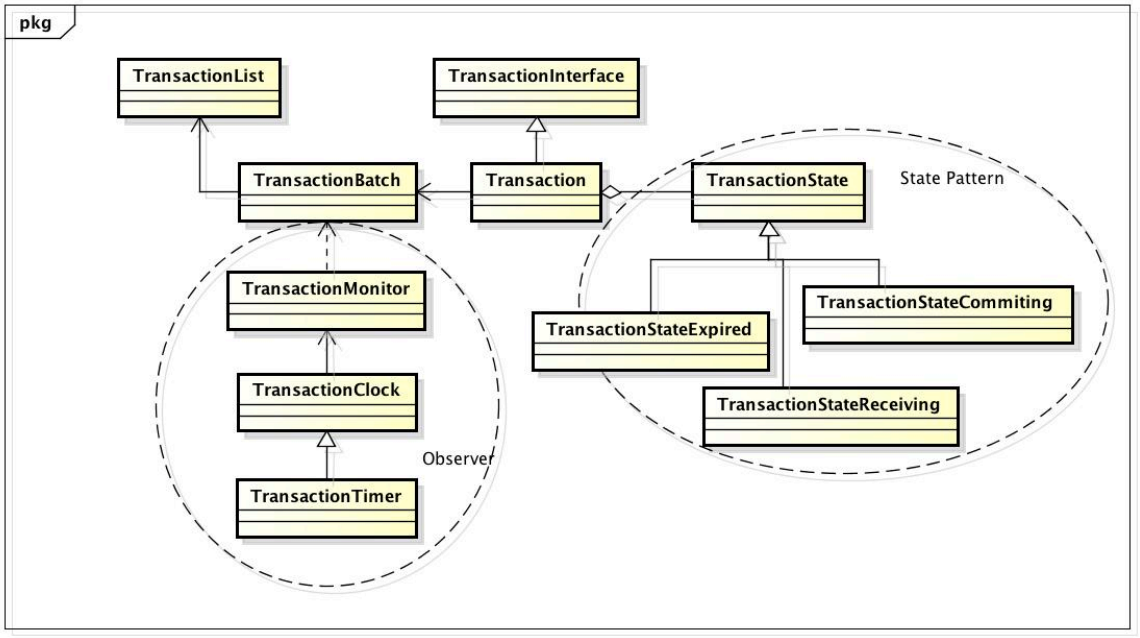


Figure 3.2.2.10.1: Transaction Management Agent

A transaction batch will be timed out if not all the constituent transactions in the batch donot arrive within a threshold period of time. For example, while processing SQL queries, all transactions in the query must arrive within system-defined threshold for the batch to be committed. Regular file upload/downloads are considered as NoSQL data, and the transactions will timeout if not all the expected packets donot arrive within user defined threshold period.

3.2.2.11 Network, Locality and Mobility Management Agent

The Network, Locality and Mobility Management Agent are responsible for providing continuity of service when a user moves around within a geographic region. A composite *Location* class that would be accessible by a LocationMapper tracks the locations. The user can access data depending on the DataAccessStrategy class. For data that is not available within the MDIC cache can be accessed through network based on the NetworkConnectivity. A high-level design overview for this component of MDIC is shown in Figure 3.2.2.11.1.

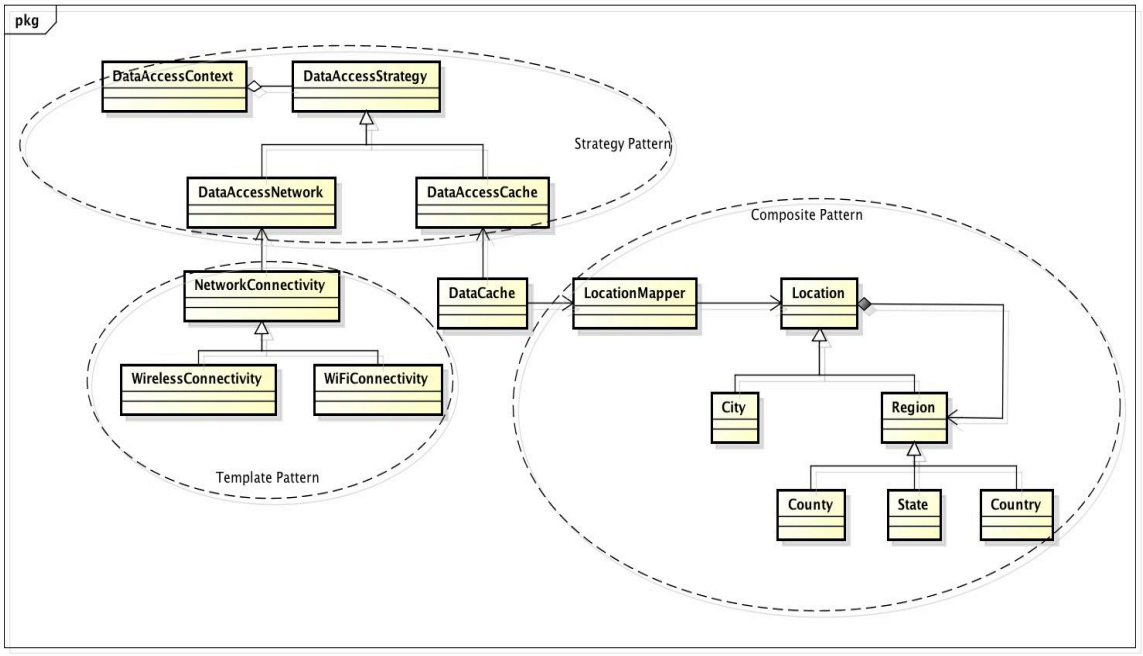


Figure 3.2.2.11.1: Network, Locality & Mobility Management Agent

3.2.2.12 Session Management Agent

The Session Management Agent is responsible for managing user sessions and maintaining connections. The Session descriptor contains the state of the session. When a client opens a new connection, the state is set to *SessionStateEstablished*. The SessionInfo associates the client information along with the Data. If the client disconnects, the state is moved to *SessionStateDisconnected*. If the client reconnects, the SessionClientInfo and SessionDataInfo are matched to restart connections. A system defined timer moves disconnected sessions to *SessionStateExpired.* SessionList is the in-core copy of the session database that is maintained at the MDaaS.

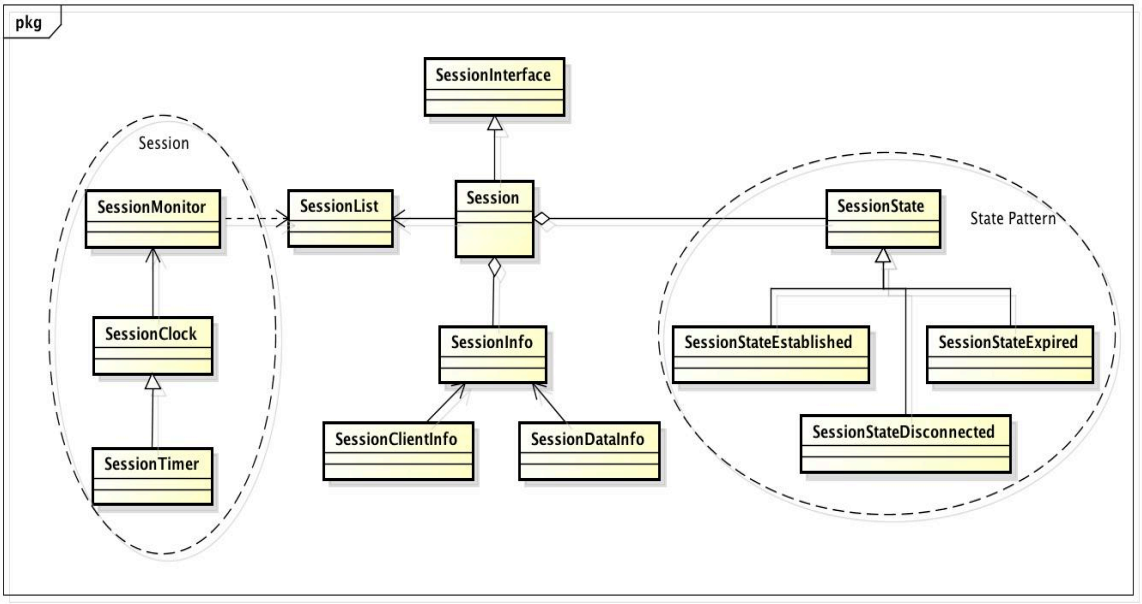


Figure 3.2.2.12.1: Session Management Agent classes

Figure 3.2.2.12.2 (an extension of Figure 3.2.2.8.2.3) illustrates Session Management control flow between the mobile client, CMDIC and RMDIC. The steps are enumerated as follows:

- During initial connection setup, mobile device (through MDaaSClient class) sends its authentication information (username/password), unique serial number, and its mac address as a parameter to getRMDICInstance web service endpoint of CMDIC.
- The getRMDICInstance will map the location of the mobile device to the nearest RMDIC and return its IP address to the MDaaSClient.
- Subsequently, an initializeDB call will be made from the mobile device that will map the device with the registered tenant, clients and any of established sessions from that device.
- A unique session object handle will be created that contains:
    - Session ID GUID
    - Client ID
    - Tenant ID derived from the <unique serial number, application ID, mac

address of the mobile unit> that has been mapped with the registered tenant of MDaaSClient.

- MDaaSClient object will save the session object handle, which will be sent across to the MDIC in future transactions.
- Session objects will be added to SessionList (SessionID-to-Device-Map-table) and valid for 1 hour.
    - New session IDs have to be reissued every hour by exchanging new session object between MDIC and client

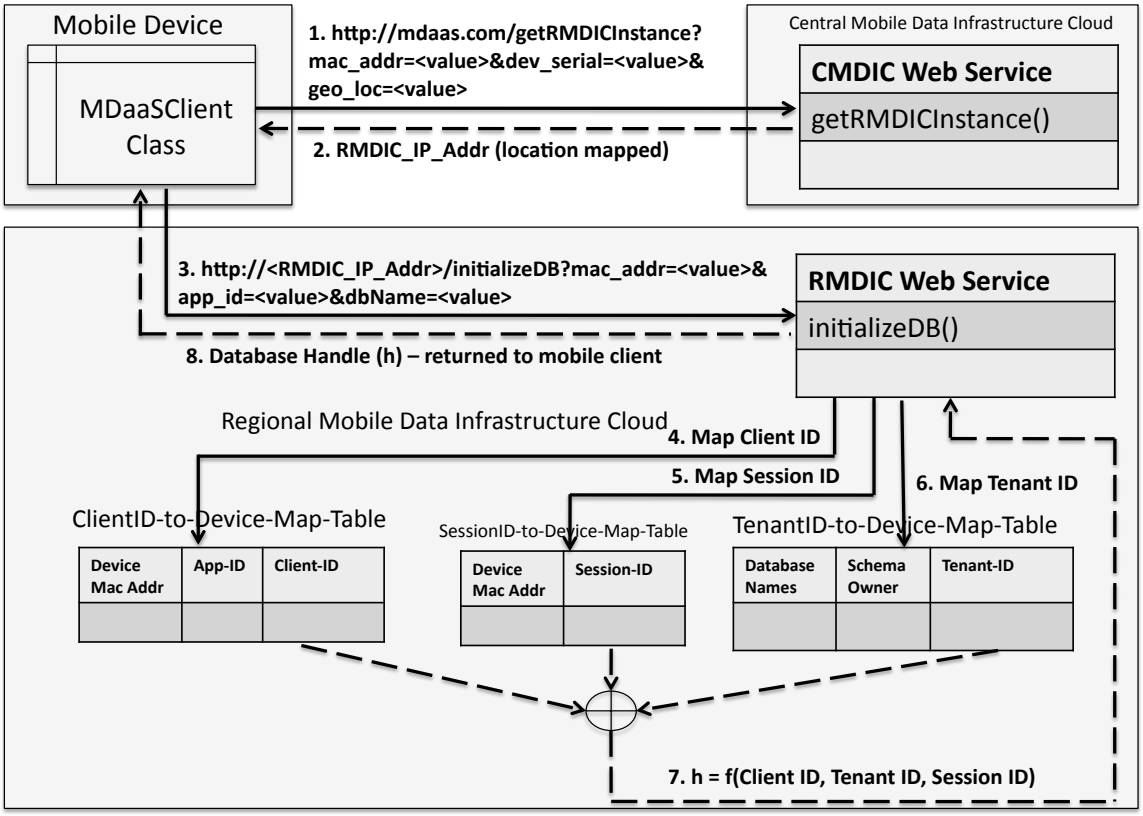


Figure 3.2.2.12.2: Session Management Control Flow

It may be noted that in first release, we may not support restarting of previously established connections. As shown in Figure 3.2.2.12.1, SessionState will depict status of the current session.

3.2.2.13 Scalability Management

This section enumerates the scalability considerations of MDaaS. The following table summarizes different scalability techniques along with a brief description on how this will be achieved.

Layer	Technique	Description
Backend Data Cloud	Storage Access Load Balancer	Zoo-keeper load balancer will make sure that database and storage is accessible seamlessly.
	BDC service scaleout	When load on databases increase, data will be served through newly spawned Virtual Machines.
Mobile Data Infrastructure Cloud	RMDIC Service scaleout	When load on an RMDIC server increases, new Virtual Machines will be spawned that will service the CMDIC requests.
	CMDIC Service scaleout	When load on the CMDIC server increases, new Virtual Machines will be spawned that will service the client requests.
	Cache replication	As load on customer data cache increases, requests may be offloaded to neighboring RMDIC depending on the locality and load on that RMDIC.

3.2.3 Mobile Client Tier

In this section we will discuss the detailed design principles of the components in the mobile client for MDaaS product.

3.2.3.1 Data and Control API Service Management

MDaaS will provide apps with a set of APIs that will enable them to access databases without knowing which database is actually being used at the backend. This will enable the apps to be portable; also, MDaaS will be able to switch between database providers in order to meet scalability and QoS requirements from time to time.

3.2.3.1.1 Control APIs

A mobile client device will have apps that will enable the tenant and/or master DBA to administer databases. With these apps, the DBAs will be able to query, create, remove or modify database schemas. The apps will interact with MDaaS through the control APIs, a summary of their functionality is discussed below:

Operation	**API**
Obtain a list of schema	https://<MDaaS>/db/_design/<schema_name>/list
Obtain information about a schema	https://<MDaaS>/db/_design/<schema_name>/info
Create a new database	https://<MDaaS>/db/_design/<schema_name>/add?<schema_file>
Remove a database	https://<MDaaS>/db/_design/<schema_name>/remove
Update a schema	https://<MDaaS>/db/_design/<schema_name>/update?operation=<add|remove>,<schema_file>

3.2.3.1.2 Data APIs

The Data APIs will be used to actually retrieve and/or store data in the databases.

Operation	API
Query database	https://<MDaaS>/db/_design/view/<database_name>/<query_statement>
Update database	https://<MDaaS>/db/_design/store/<database_name>/<update_statement>
Insert database	https://<MDaaS>/db/_design/store/<database_name>/<insert_statement>
Delete database	https://<MDaaS>/db/_design/store/<database_name>/<delete_statement>

3.2.3.1.3 Sample Database Queries

Select * from PropertyOnSaleTable where zipcode=95014 and price <= 600,000

From mobile application, query syntax will be as follows:

```
mdaasClient.queryDB(String tableName, ArrayList<ArrayList<String,
String>> filter)
      where, tableName = ”PropertyOnSaleTable”
      filter[0] = {“zipcode”, 95014}
      filter[1] = {“price”, 600,000}
```

3.2.3.1.4 Sample Database Updates

Update PropertyOnSaleTable set price = 610,000 where propertyID = 10272

From mobile application, update syntax will be as follows:

```
mdaaSClient.updateDB(String tableName, String keyName, String keyValue,
ArrayList<ArrayList<String,String>> updateParams)
      where, tableName = “PropertyOnSaleTable”
      keyName = “propertyID”
      keyValue = “10272”
      updateParam[0] = { “price”, 610000 }
```

Similar considerations for “insertDB” and “deleteDB” operations from table

3.2.3.1.3 Informational APIs

For querying and exchanging data with MDaaS servers, clients will use a language that is similar to what is shown in Section 3.2.3.1.2. Clients will contact MDaaS servers by

using special connection APIs that is discussed in Section 3.2.3.2. This request will land in the CMDIC, which will forward it to the appropriate RMDIC. CMDIC will respond to the client with the IP address of the RMDIC. Once the right RMDIC is identified, subsequent connections will be directly made with RMDIC itself.

Operation	**API**
Get RMDIC instance	https://<MDaaS>/getRMDICInstance
Query RMDIC Information	https://<MDaaS>/db/_design/view/<rmdic_name>/info

3.2.3.2 Mobile Client Connectivity Manager

This component is responsible for maintaining connectivity with the MDaaS layer. This is the lower or outer most layer in the mobile device that talks to the MDaaS. The connection is established with CMDIC server, which upon authentication of the mobile device, will send the detail about the RMDIC that will actually service the connection. Subsequently the connection will be established with the RMDIC. The Mobile Client Connectivity Manager follows Strategy Design Pattern as shown in Figure 3.2.3.2.1.

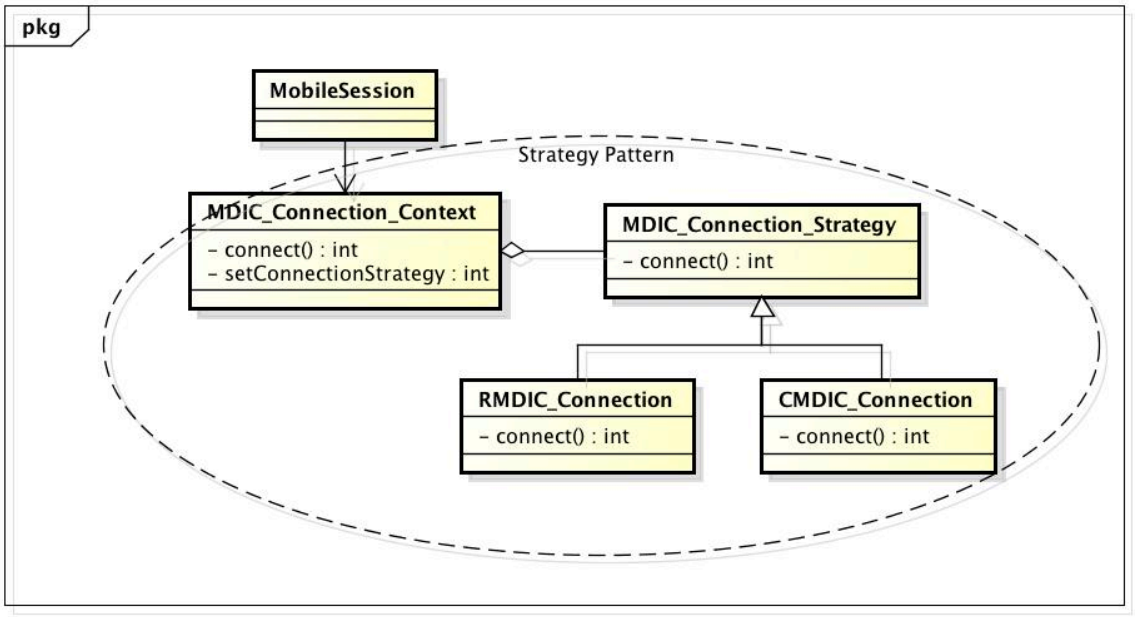


Figure 3.2.3.2.1: Mobile Client Connectivity Manager

For connectivity, mobile clients will be able to use RESTful services or Java APIs that are similar to JDBC connection requests. Some sample APIs are enumerated in the table below:

Operation	API
Connect to database	https://<MDaaS>/db/_design/connect/<database_name>
Query database list	https://<MDaaS>/db/_design/connect/db_list?org=<organization_name>

3.2.3.3 Mobile Client Session Manager

This section describes the design of Mobile Client Session Manager. This module establishes and maintains the sessions with the MDIC. Once a session is established, the MDIC sends a cookie to the mobile client, which is valid for a prescribed period of time. Once the session cookie expires, the mobile client must request a fresh cookie. The SessionMonitor gets triggered by expiry of the SessionTimer clock, which results in requesting for the new cookie. The state of the session is maintained in MobileSessionState.

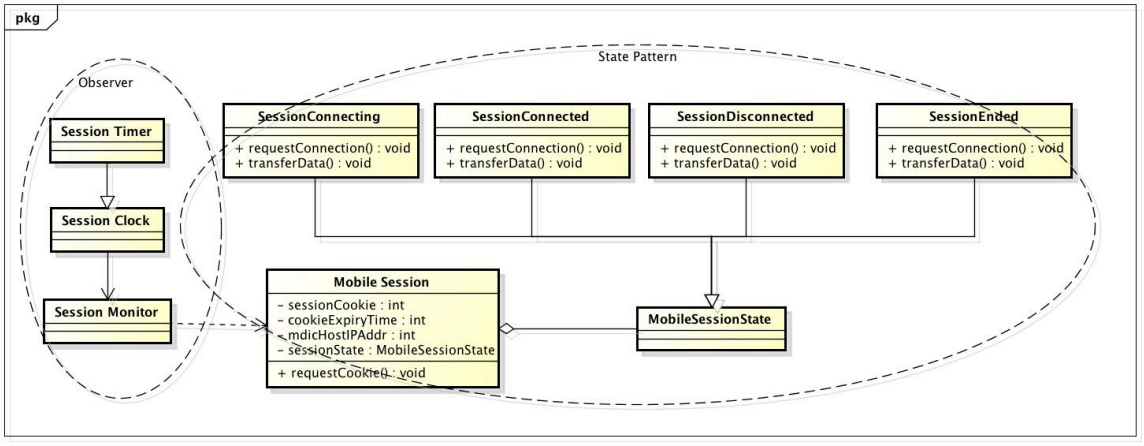


Figure 3.2.3.3.1: Mobile Client Session Manager

Session expiry ways:

(a) Explicit logout by the app (someone closes the app)
(b) Idle/disconnected state of the client device from server for an extended period of time.

Reconnecting to server after both of the above cases will require exchange of new session ID between the mobile device and server.

3.2.3.4 Client Software Architecture

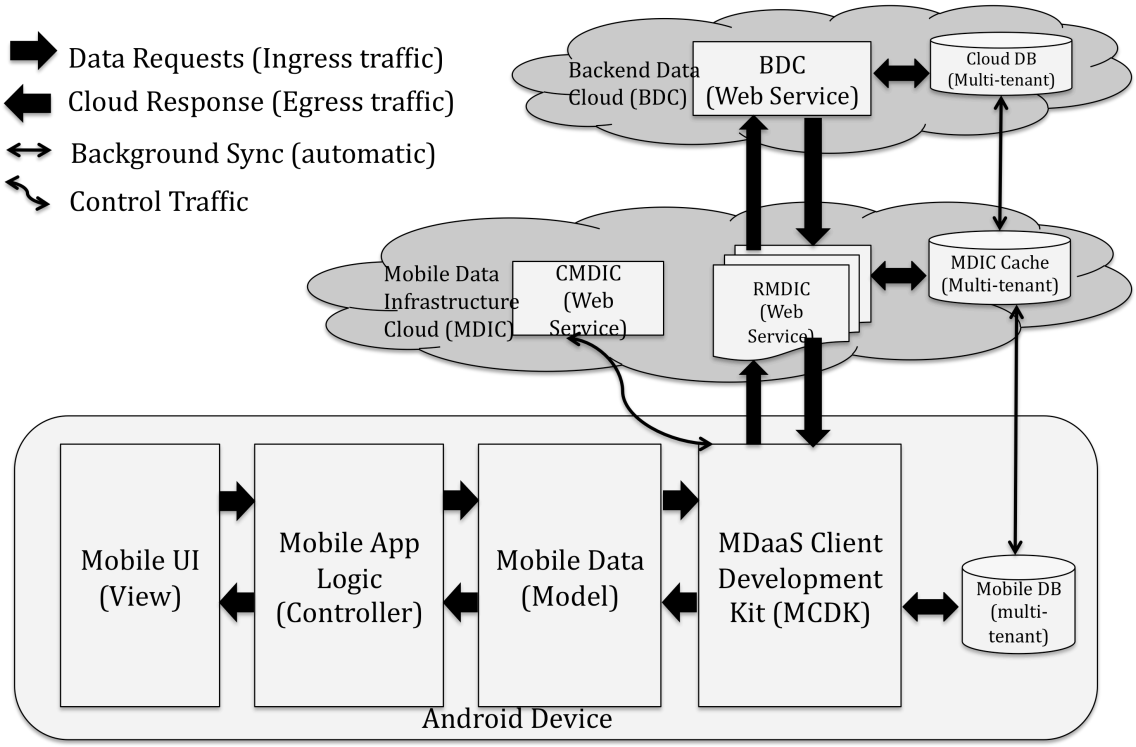


Figure 3.2.3.4.1: Mobile Client MVC Architecture

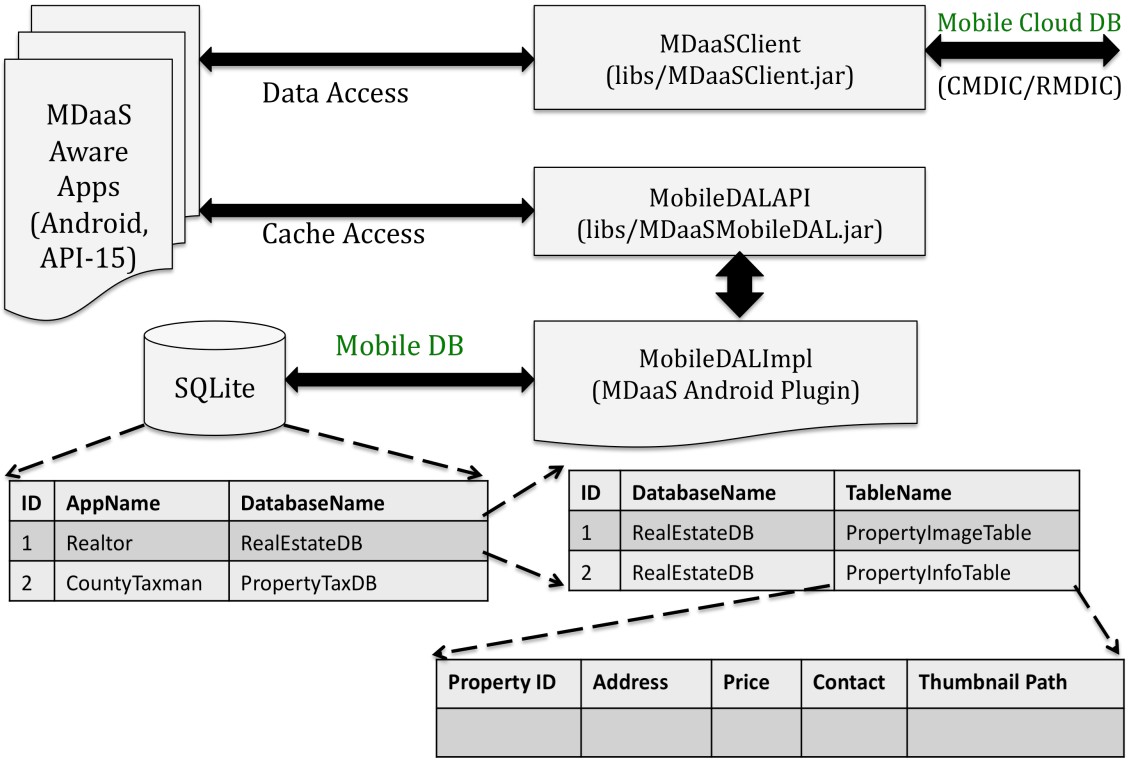


Figure 3.2.3.4.2: MDaaS Client Development Kit (MCDK)

Content Provider URI	Description
content://com.mdaas.mobility.provider.DataManager/db	Returns information about all known databases in the client.
content://com.mdaas.mobility.provider.DataManager/db/1 content://com.mdaas.mobility.provider.DataManager/db/2	Returns metadata info for a specific database identified by #.
content://com.mdaas.mobility.provider.DataManager/db/create? dbName=<db>, appName=<app>	Create a database
content://com.mdaas.provider.DataManager/db/1/table	Returns information of all tables for a given DB identified by #
content://com.mdaas.provider.DataManager/db/1/table/1 content://com.mdaas.provider.DataManager/db/1/table/2	Returns (schema) information of a specific table identified by #.
content://com.mdaas.provider.DataManager/db/1/table/1/create? table=<tableName>,colList=<colName,colType>[,...]	Create a table with a number of columns in database identified by #
content://com.mdaas.provider.DataManager/db/1/table/1/insert? {colList=<colName,colValue>[,...]}	Add a row in a table identified by #
content://com.mdaas.provider.DataManager/db/1/table/1/query? {<SQL statement>}	Query data from a table identified by #

3.3 System Interface and Connectivity Design

This section discusses the system infrastructure and connectivity design of MDaaS. The users' mobile devices communicate with the MDaaS through the wireless medium either through wireless carrier (3G/4G) or through their ISP (broadband or WiFi). Once the database access request reaches the MDaaS layer, it is routed to the respective RMDIC server for the user, wherefrom it is routed to the BDC through high-speed Internet. Figure 3.3.1 demonstrates the connectivity and relationship between various services of MDaaS. Table 3.3.1 provides some detail about individual servers that are involved in the whole system.

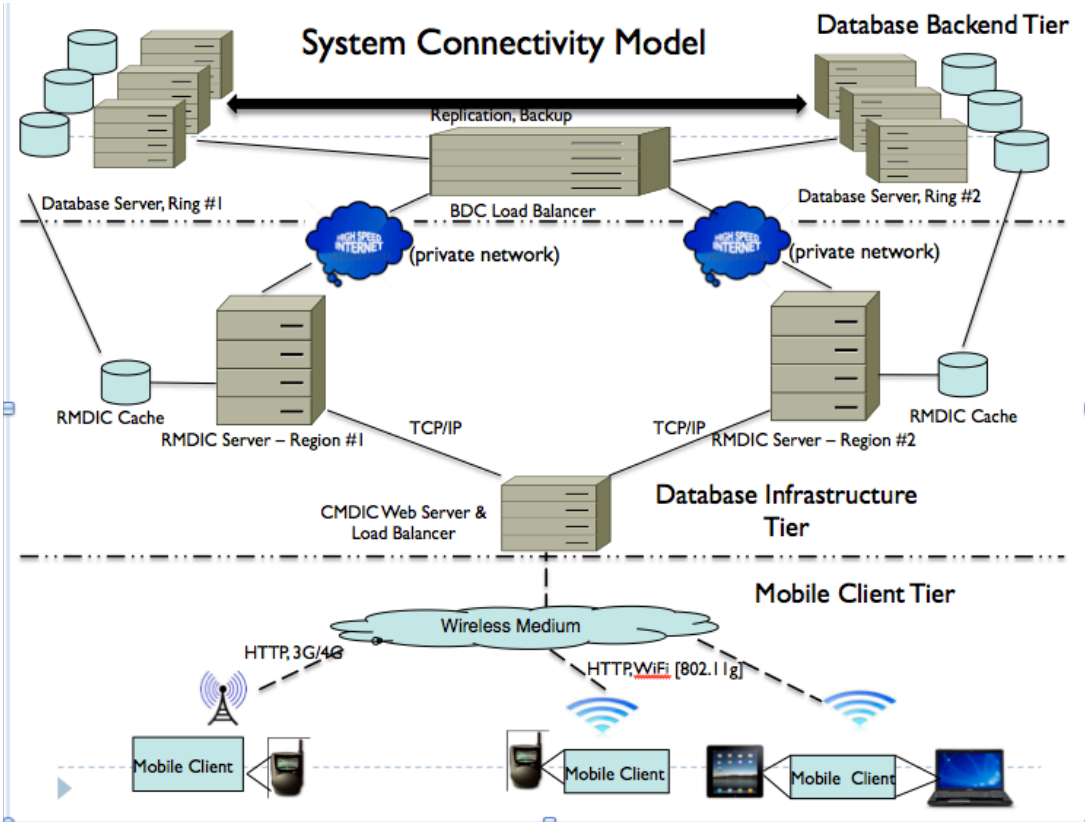


Figure 3.3.1: Infrastructure & Connectivity Model of MDaaS

Table 3.3.1: MDaaS Infrastructure & Connectivity Model

System Name	Description
Mobile Clients	Mobile clients constitute the devices wherefrom an end-user initiates connections with the MDaaS. They communicate via WiFi or Wireless media to the CMDIC via cloud gateway server and routers using HTTP protocol.

CMDIC Web server and load balancer	This server receives the web service requests from the mobile clients. Here tomcat web server redirects the requests to the corresponding backend classes, which communicate to the corresponding RMDIC servers.
RMDIC Server	RMDIC servers receive the requests from the CMDIC, which actually implements various QoS and MDaaS functionality, and forwards them to the BDC.
RMDIC Data Cache	RMDIC data cache nodes are associated with the RMDIC servers and are used to respond to requests whose data are locally available.
BDC Load Balancer	RMDICs communicate with the BDC through the BDC load balancer servers.
BDC DB Server	BDC DB servers enable accesses to the actual databases resident in the cloud.

3.4 System Data and Database Design

This section examines some of the system level (control) data structures that will be maintained by MDaaS. These data will be maintained by MDaaS as control information, and will be used at different points of its operation.

3.4.1 CMDIC Database

3.4.1.1 CMDIC-RMDIC-Mapper

This table will be maintained in the CMDIC and will be used to map and route a user request to the corresponding RMDIC.

Column Name	Type	Description
RMDIC_ID	String	Unique ID of RMDIC
RMDIC_Location	{String,String}	Co-ordinates (geo-location) of the RMDIC. This will be used to map the user request to their nearest RMDIC so that subsequent requests in that connection will be handled directly by the RMDIC.
RMDIC_IPAddr	String	IP Address of the RMDIC
RMDIC_MaxConnections	Integer	Max number of concurrent connections that can be addressed by the RMDIC. If the number of active RMDIC connections exceed the max number of connections, subsequent requests will be diverted to the nearest available RMDIC.
RMDIC_ActConnections	Integer	Current number of active connections that are being addressed by the RMDIC.

3.4.2 RMDIC Database

3.4.2.1 RMDIC-Client-Mapper

This table will be maintained in the RMDIC and will be used to maintain the list of mobile clients that are connected to it.

Column Name	Type	Description
RMDIC_ID	String	Unique ID of RMDIC

Client_IPAddr	String	IP Address of the client device
ConnectionStartTime	String	Start time of the connection
ConnectionEndTime	String	End time of the connection
TransferBytes	Integer	Amount of data transferred (uplink/downlink)
TransferStatus	Boolean	Status of transfer

3.4.2.2 RMDIC-BDC-Mapper

This table will be maintained at each RMDIC in order to map Backend Databases.

Column Name	Type	Description
RMDIC_ID	String	Unique Identifier of the RMDIC
BDC_ID	String	Unique Identifier of the BDC
BDC_Provider	String	Name of the Provider of the BDC
BDC_IPAddr	String	IP Address of the BDC

3.4.2.3 RMDIC-CMDIC-Mapper

Column Name	Type	Description
RMDIC_ID	String	Unique Identifier of the RMDIC
CMDIC_ID	String	Unique Identifier of the CMDIC

3.4.2.4 RMDIC-BDC-Connections

Column Name	Type	Description
RMDIC_ID	String	Unique Identifier of the RMDIC
BDC_ID	String	Unique Identifier of the BDC
ConnectionID	String	Unique identifier representing the connection

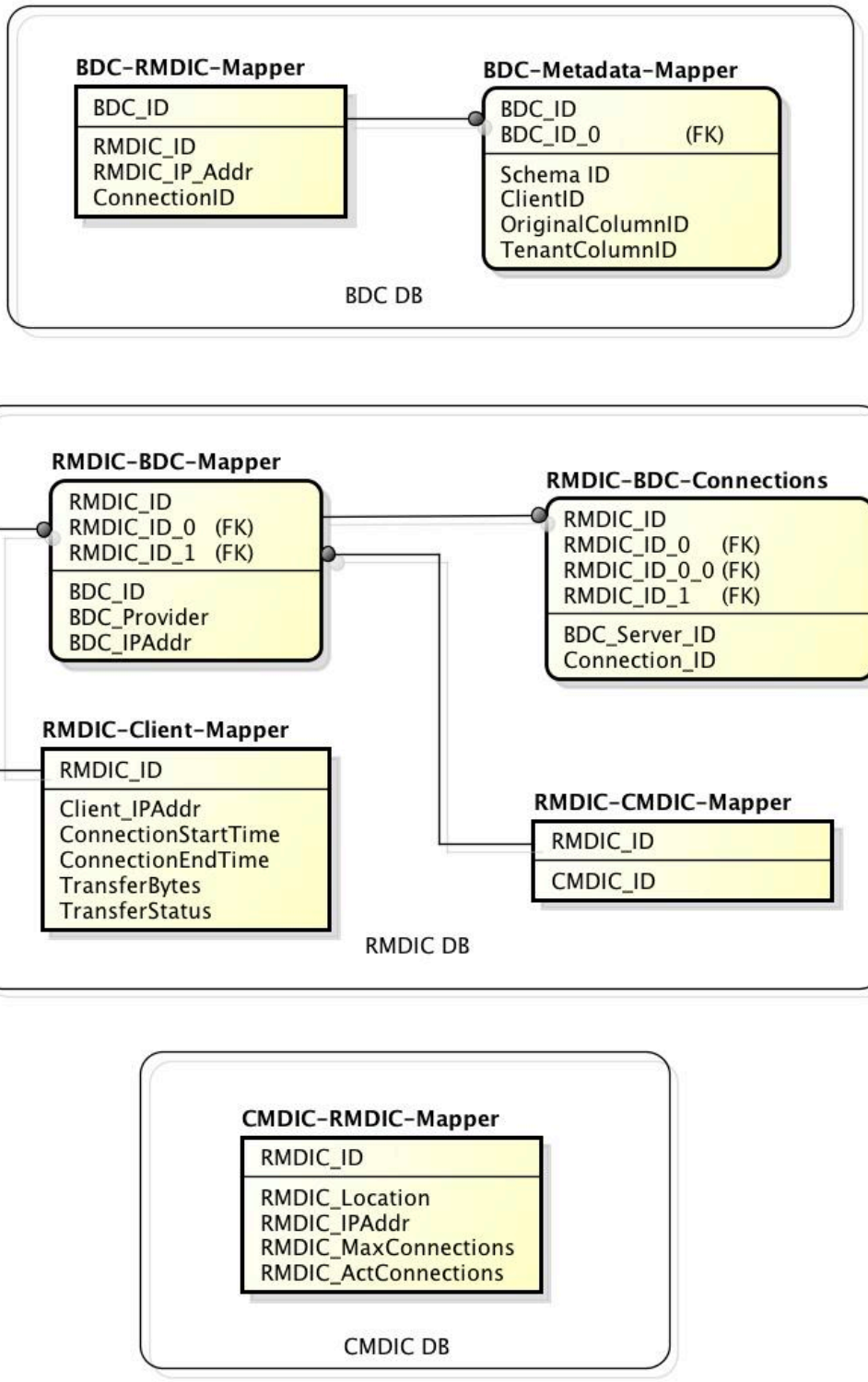
3.4.3 BDC Database

3.4.3.1 BDC-RMDIC-Mapper

This table will be maintained at the BDC to keep track of the RMDICs that connect to it.

Column Name	Type	Description
BDC_ID	String	Unique ID of the BDC
RMDIC_ID	String	Unique ID of the RMDIC
RMDIC_IPAddress	String	IP Address of the RMDIC
ConnectionID	String	Unique ID representing the connection

3.4.4 Control DB – Entity Relationship Diagram



3.5 User Interface Design

3.5.1 Mobile User Management UI

This portal is responsible for various management functionalities depending on the class of user:

- *Regular user*: A regular user is the end-user who would interact with the system for the following purposes:
  - Registration
  - Credential management (password administration, etc)
- *Tenant DBA*: A tenant DBA is responsible for managing a set of regular users typically from a company. They would be accessing the portal for the following purposes:
  - Schema administration
  - Access control administration for users
  - Initiating backups and restores
- *Master DBA*: The master DBA has control over the whole system. They would typically access the system for the following reasons:
  - Access control administration for all tenants
  - Meta-data management
  - Initiating backups and restore for all tenants if necessary
  - Maintenance of the master schema

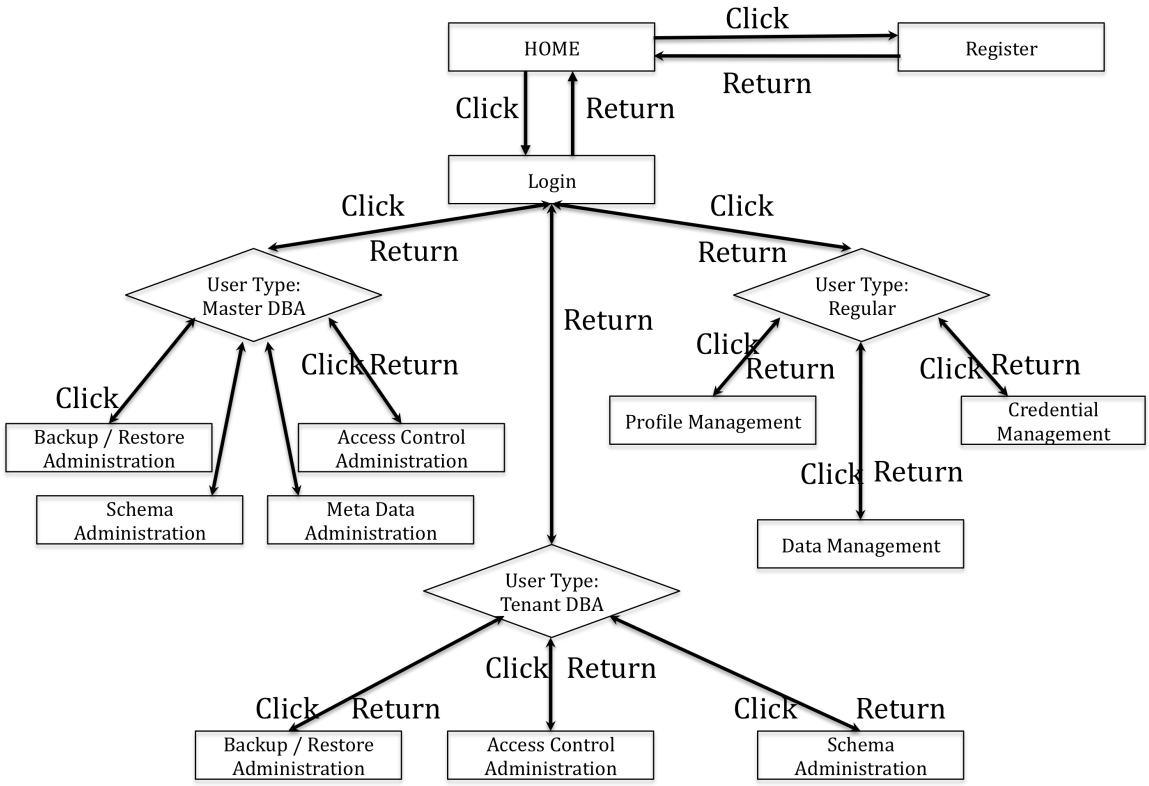


Figure 3.5.1.1: Mobile User Management UI

3.5.2 MDIC Administration UI

This portal is responsible for administration of the Mobile Data Infrastructure Cloud (MDIC). Both the tenant and the master DBAs are allowed access to this portal. The key functionalities that can be performed from this portal are as follows:

- *Provisioning of RMDICs*
  - Orchestration of RMDIC node
    - Spawning of VMs that will serve the mobile client requests, etc
    - Imaging / Software Upgrade of RMDIC node and VMs
    - Status health checking of nodes
  - Schema management
  - Customer data management (replication from BDC)
- *Configuration of RMDICs*
  - Managing Load Balancing Configuration
  - Managing storage cache
- *Resource Management*
  - Tracking of resource usage from each user per tenant
  - Billing

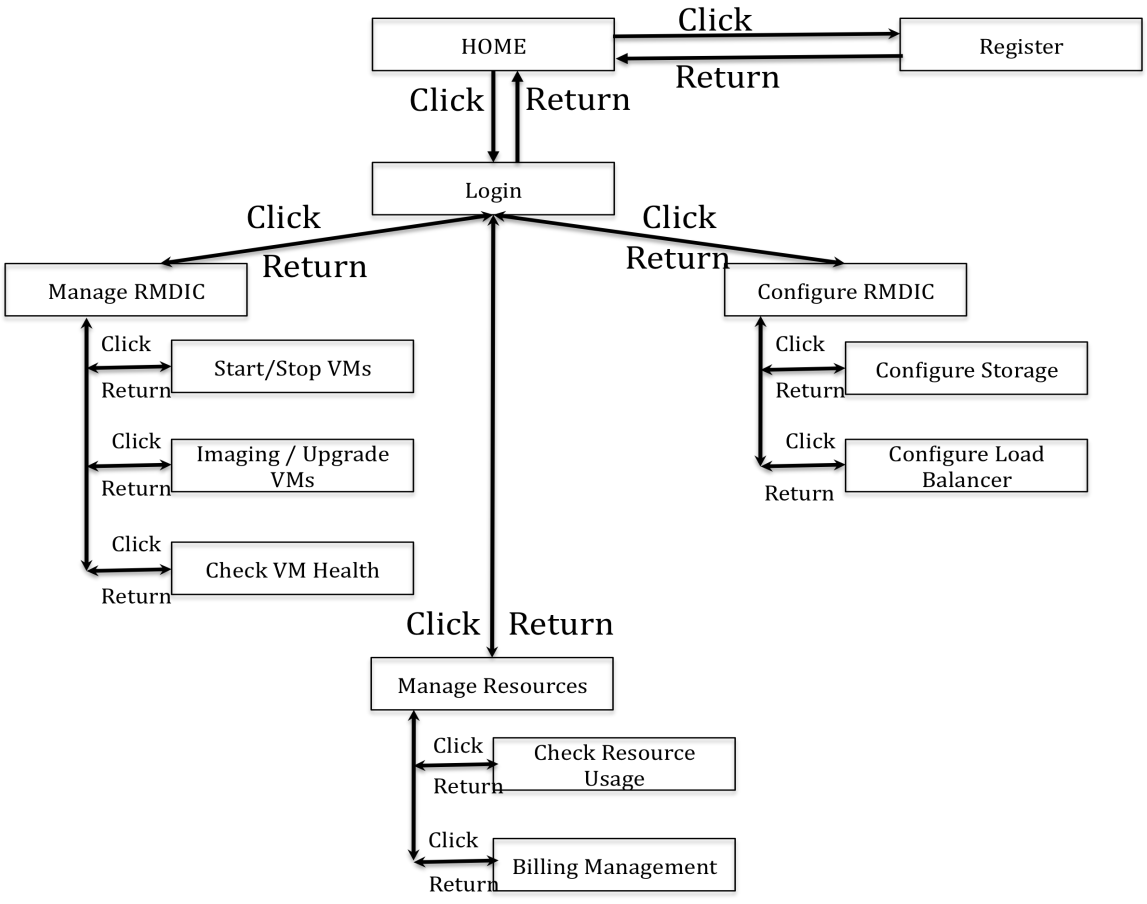


Figure 3.5.2.1: MDIC Administration UI

3.5.3 BDC Administration UI

This portal is responsible for administration of the Backend Data Cloud (BDC). Only the master DBA will be allowed access to this portal. The following tasks can be performed from this UI:

- Provisioning of BDCs
  - Orchestration of BDC node
    - Spawning of BDC VMs that serve the backend databases
    - Imaging and software upgrade of BDC host and VMs
    - Status or health check of nodes
- Backup management
- Disaster recovery management
- Configuration management
  - Database tuning
  - Backend storage management

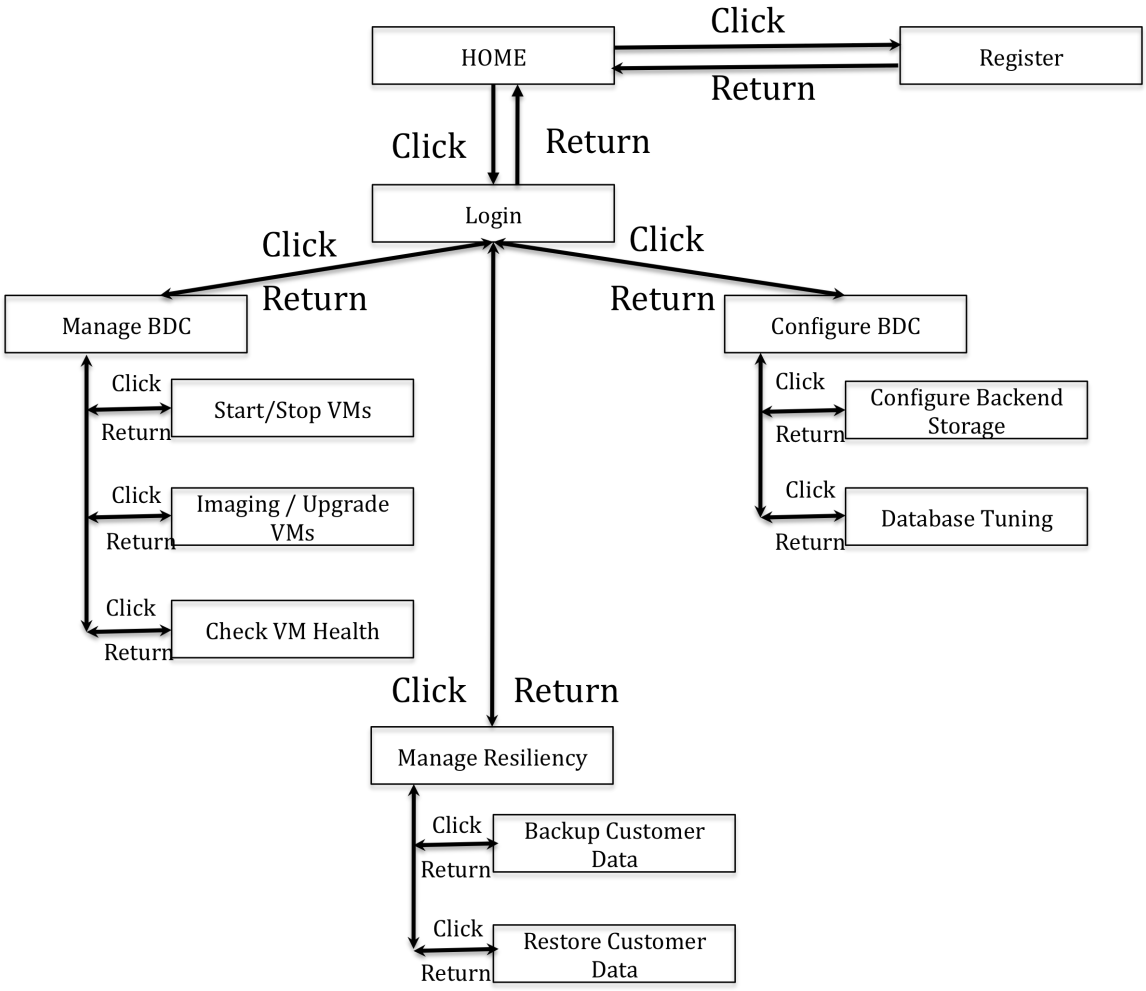


Figure 3.5.3.1: BDC Administration UI

3.6 Design Trade-offs and Solutions

Wireless networks present with unique design challenges for MDaaS that must be taken into consideration. Some of these challenges are summarized below:

- Traffic patterns, network bandwidth and load, and location of end-users change continuously and even in the middle of a transaction.
- Constraints about low-power and energy capacity of the devices and latency/delay induced to transactions due to relatively slower medium (unlike Gigabyte Ethernet) pose newer challenges to the design across all layers of the software.

Various trade offs for design of MDaaS were examined in order to make sure that the system would scale appropriately even in the face higher demand and load when it is placed in production. We envisage that the target environment for MDaaS would be both WLAN and WWAN. The remainder of this section discusses the various design trade-offs and their respective solutions.

3.6.1 Design Trader-off #1: MDIC caching v/s direct BDC access

MDaaS sits in between the mobile database clients and the distributed master databases residing in the cloud. However, access latencies can be improved significantly if every data access does not result in an access to the cloud. Two possible design strategies are possible:

- *Direct BDC access for each transaction*: In this approach, all data accesses require interacting with the cloud. Such a strategy results in latency with each transaction.
- *Caching data at RMDIC*: In this approach, database reads result in searching the RMDIC cache, while writes result in updating the cache at the RMDIC. As a user moves between RMDICs, the contents of the cache are transferred to the new RMDIC where the user moves.

It is thought of that caching data at RMDICs will result in better performance and enhance customer experience. It will however require provisioning of storage at each RMDIC, which will require its own maintenance.

3.6.2 Design Trader-off #2: Load Balancer strategy selection

Load balancers are required when a mobile unit talks to the MDaaS as well as when MDaaS talks to the BDC. Various Load Balancer strategies are possible for implementation:

- *Full-NAT approach:* In this approach both the destination as well as source IP addresses are rewritten as a packet traverses through the load balancer
- *Half-NAT approach:* In this approach only the destination IP address is changed as the packet traverses through the load balancer.

Type	Direction	Half NAT		Full NAT	
		Source IP Address	Destination IP Address	Source IP address	Destination IP address
Request	Client -> LB	Client	VIP of LB	Client	VIP of LB
	LB -> Server	Client	Server	Interface address of LB (subnet C)	Server
Response	Server -> LB	Server	Client	Server	Interface address of LB (subnet C)
	LB -> Client	VIP of LB	Client	VIP of LB	Client

- *Direct Server Return approach:* In this approach, the Load Balancer is out of the outgoing path, and as a result, this more scalable than Full/Half NAT approaches.

Type	Direction	DSR		
		Source IP Address	Destination IP Address	Destination MAC Address
Request	Client -> LB	Client (1.1.1.1)	VIP of LB (2.2.2.2)	MAC of LB (AAAA)
	LB -> Server	Client (1.1.1.1)	VIP of LB (2.2.2.2)	MAC of Server (BBBB)
Response	Server -> Client	VIP of LB (2.2.2.2)	Client (1.1.1.1)	MAC of default gateway

3.6.3 Design Trade-off #3: Determining Capacity of RMDIC Servers

Bandwidth requirement for each connection is a crucial design consideration as it determines how much data a mobile device is likely to use. Once this information is available, it can be used to determine the total bandwidth for the whole area. This information will be useful to determine the capacity of MDaaS servers that need to be deployed in an area. Figure 3.6.3.1 displays a graphical representation of typical bandwidth requirement data that was collected for mobile connections using WLAN [56].

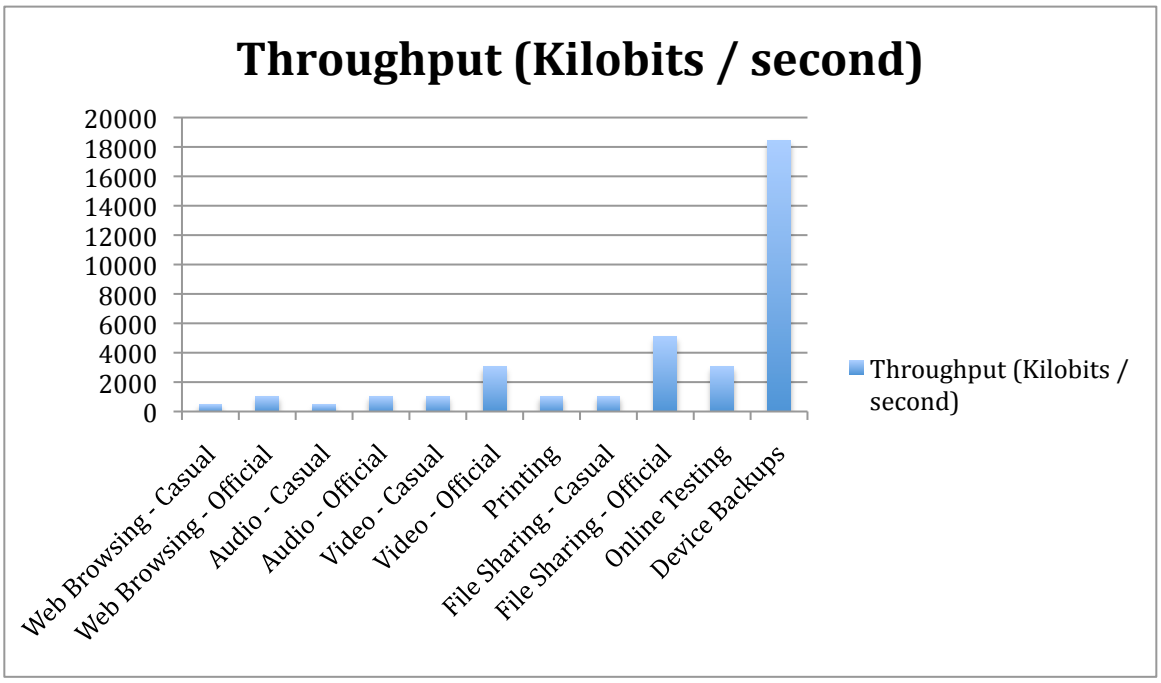


Figure 3.6.3.1: Throughput v/s Application type – RMDIC server capacity [56]

3.6.4 Design Trade-off #4: Determining number of RMDICs for an area

Aggregate throughput requirement for an area will determine how many MDaaS servers will be required for that location. For a densely populated area, the aggregate throughput is expected to be higher, which directly translates to the number of RMDICs that need to be provisioned in an area. This partitioning will help in more reliable connectivity and better QoS, which directly translate to more customer satisfaction.

The number of connections in a cell is what determines the total throughput that will be realized per connection. As most users today carry more than one computing device (such as a tablet computer, laptop, or smartphones), they contribute directly to the network resources and are part of aggregate bandwidth calculation. An increase in numbers of device connections is one of the primary reasons older WLAN designs are reaching oversubscription today.

As seen in Figure 3.6.4.1, the throughput for various types of WiFi connections reduce when no of clients increase. As a result, in highly populated areas, larger number of RMDICs must be provisioned.

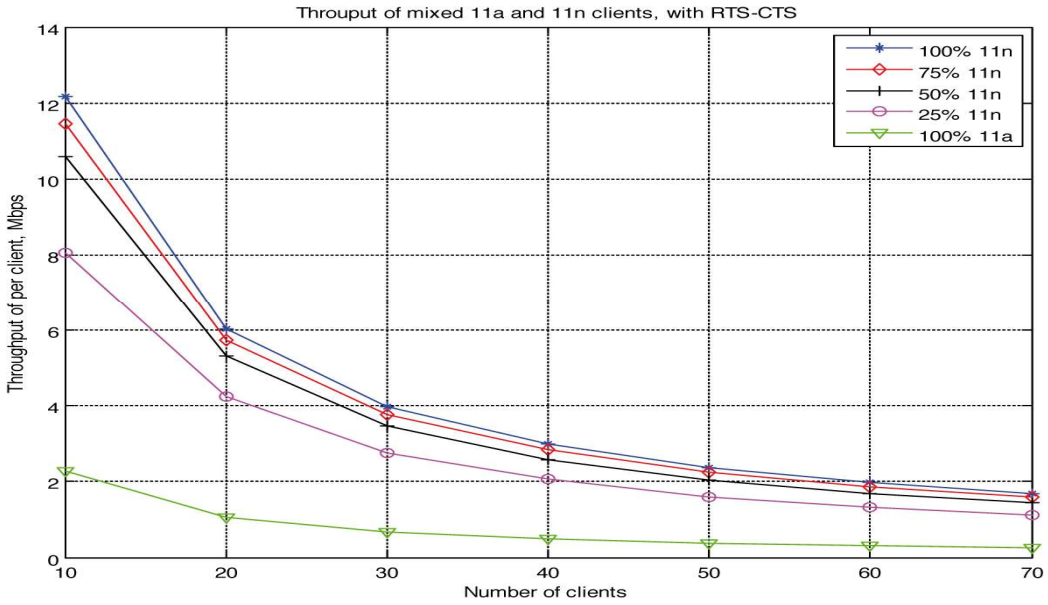


Figure 3.6.4.1: Throughput v/s No of Clients [56]

3.6.5 Design Trade-off #5: Thin-Client v/s Smart-Client

Thin clients are mobile devices that have lesser processing power whereas modern day smart-phones or tablet computers fall under the category of smart clients. Thin clients are typically used for voice-only purposes, while smart-phones are used for both voice and data. As data accessibility is prime target of this product, smart-phones are the prime target users. As discussed in earlier sections of this chapter, some of the components of the design will be required in the mobile end-devices as well.

3.6.6 Design Trade-off #6: Data storage in mobile clients

Two possible design approaches are possible when it comes to storing data in the mobile clients. Traditional mobile databases typically cache some data in the devices themselves in order to reduce access latency. This requires frequent need for synchronization of data between the end devices with the master DB thereby increasing data bandwidth requirements. In MDaaS, data cache location will be moved from mobile device to the corresponding *Regional Mobile Data Infrastructure Cloud* corresponding to the user location. This will serve the following benefits:

- Reduce the storage need in the end-devices
- No requirement to frequently synchronize data between the master database and the end device, which directly translates to more efficient utilization of bandwidth
- As RMDIC will cache data, it will enable better mobility services depending on the location of the user.

3.7 System Deployment Model

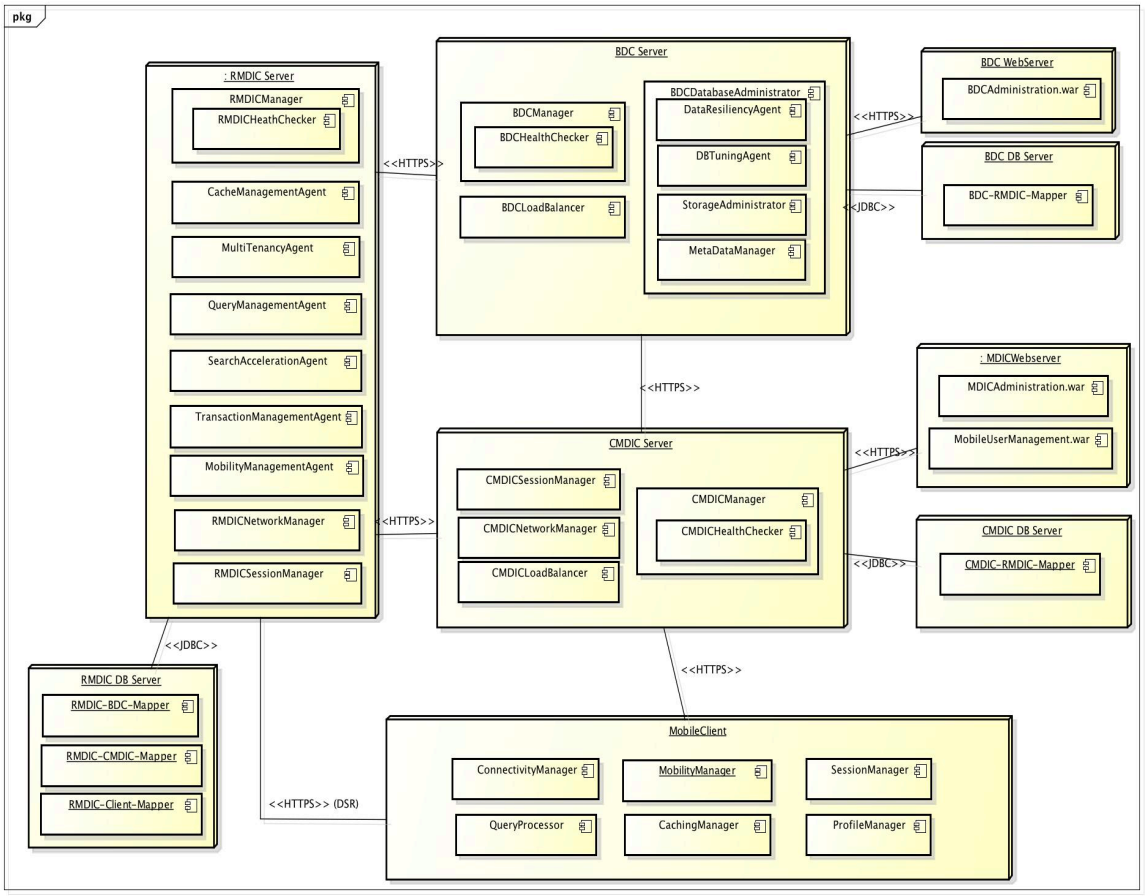


Figure 3.7.1: MDaaS Deployment Model

3.8 System Security Design

Standard security algorithms do not work in wireless environment because of the following reasons:

- They have too much overhead and tight timeout
- They are too much complex and needs much for power, storage
- They are not efficient enough for wireless networks and mobile devices
- Security policy enforcement and implementation depends on the carrier
- Security features differ between different wireless protocol stacks

Table 3.8.1 discusses some of the common threats and their remedial measures that would be taken in MDaaS.

Table 3.8.1: MDaaS Security Design

Threat Name	Type	Description of attack / threat	Remedy
Eavesdropping	Passive	Eavesdropper can listen to a message without altering the data, the sender and intended receiver of the message may not even be aware of the intrusion	MDaaS will use session ID along with the MAC address of the connecting device. If an eavesdropper connects later, their MAC address will be different and MDaaS will terminate the session. Install certificates in mobile host and CMDIC server so that trusted bond is established between the two prior to sending sensitive information. Example: this mechanism is adopted by Verisign
Unauthorized access	Active	An intruder can access the database as an unauthorized user. Once inside, they can send, receive, alter, access or forge data	Allow only authenticated users enter the MDaaS system.
Denial of Service attack (Interference and jamming)	Active	If an attacker has a powerful automated tools, they can generate too many requests that the MDIC servers become so busy that they cannot handle genuine requests.	Use TCP-SYN cookies in the OS implementation that is selected for the MDIC nodes.
Man-in-the-middle-of-	Active	The attacker will intercept the connection when a user initiates	Same as the remedy for eavesdropping (refer

attack		a connection, and complete the connection to the intended resource and proxy all communication to the resource. Now, he can modify, eavesdrop, and inject data on a session.	above).
Data Integrity	Active	Data integrity is another important concern for wireless network security. Packets of data sent over the network may be intercepted and modified in transit. For example, wireless payment system needs a solution at the payment server to check the received payment information from the client has not been changed.	A secret key scheme can be used to generate a message authentication code (MAC) that protects sensitive information (like passwords, etc) from malicious changes. MAC is a fixed-length value, known as a message integrity code.

3.9 The Big Picture

Figure 3.9.1 and Figure 3.9.2 illustrates a converged end-to-end detailed class design of Mobile Data as a Service product. The server side class design involves integration of all the classes that were discussed in Section 3.2.2, while the client side class design is a converged diagram for the design discussed in Section 3.2.3.

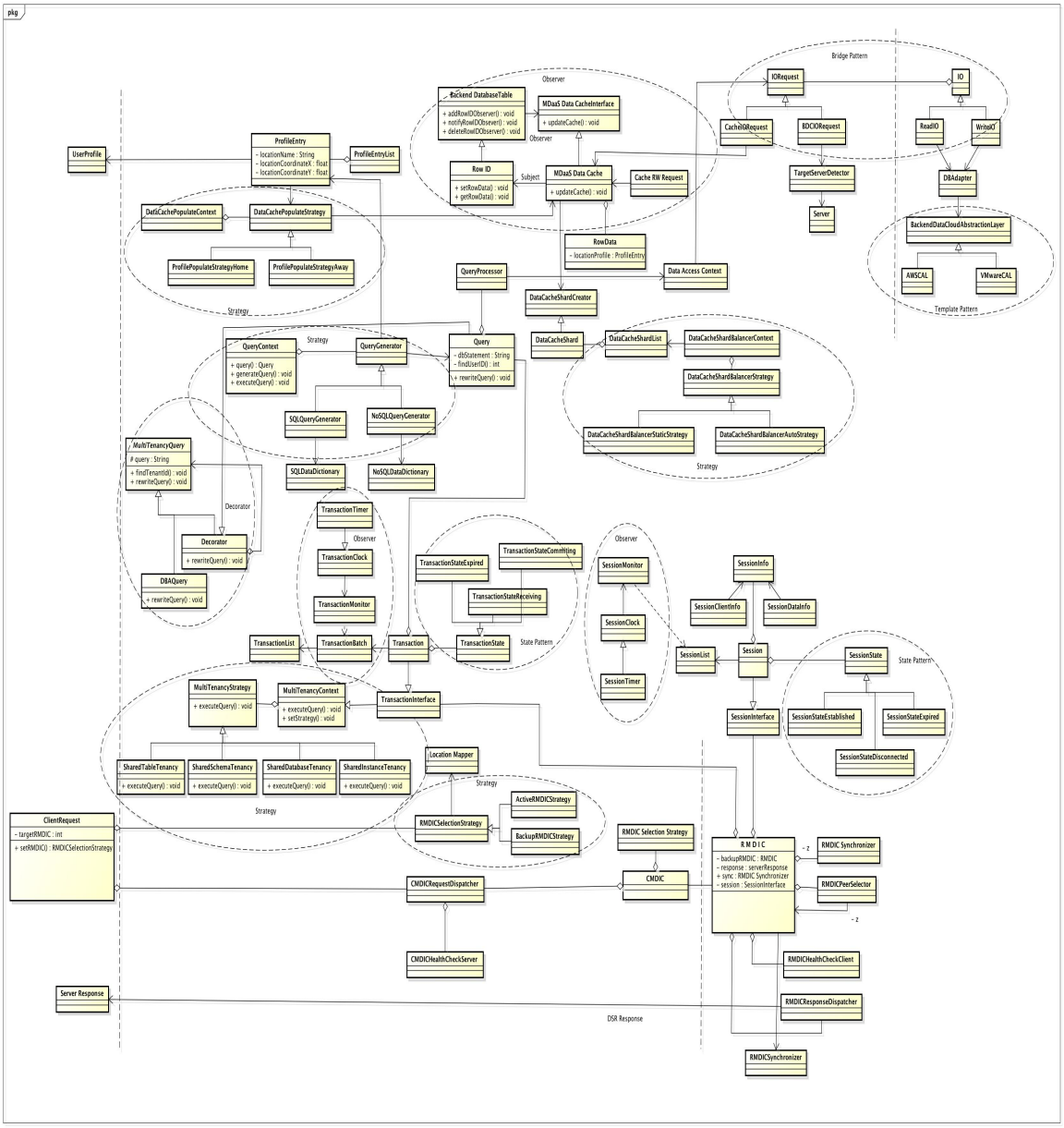


Figure 3.9.1: Big Picture for MDaaS Server

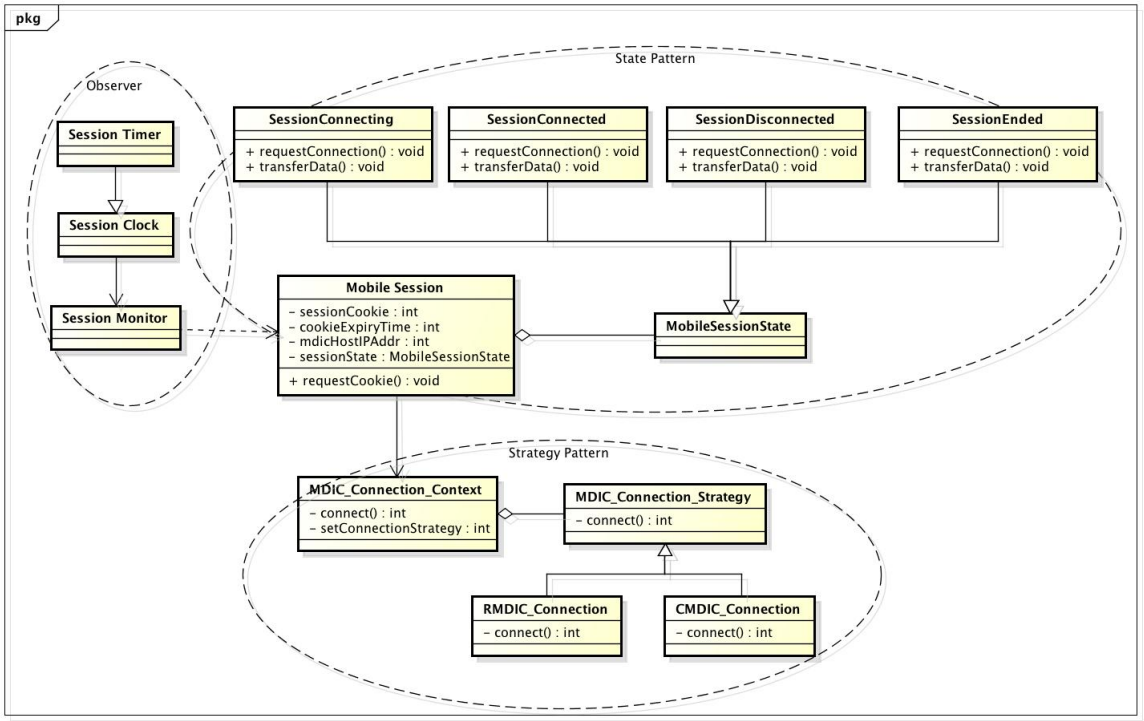


Figure 3.9.2: Big Picture of MDaaS Client Design

3.10 Mobile DBA Administration

A design for the user interface for various Mobile Database Administrative tasks was illustrated in Section 3.5. In this section we enumerate the actual tasks for Mobile Database Administration that would be supported by MDaaS. Mobile DBA administrative tasks can be classified into two main categories, viz. administration of customer data and administration of MDaaS itself. Furthermore, administration of MDaaS can be categorized into administration of *Backend Database Cloud* and administration of *Mobile Data Infrastructure Cloud*. Also as highlighted in Section 3.5, there will be two types of DBAs from their responsibility point of view: tenant DBA (responsible for administering tenant data) and master DBA (responsible for administering the MDaaS).

3.10.1 Customer Data Administration

Customer data administration involves performing a set of services like database schema management, database sharding, managing access control lists, meta-data management, multi-tenancy management and so on. Various functional components of MDIC will support these functionalities in different forms, which will be front-ended by the Mobile User Interface administration user interface as described in Section 3.5. A summary of the administrative tasks in shown in Table 3.10.1.1.

Table 3.10.1.1: Mobile Data Administration Tasks

DBA Task	DBA Type	Description
Schema Design	Tenant DBA	Each Tenant DBA will propose a schema of different databases that suits the needs of their organization.
Multi-tenancy Design	Master DBA	Master DBA will have a means to accept or customize tenant database schema proposals made by Tenant DBA based on the design proposed in Section 3.2.1.2.
Managing Access Control Lists & Security	Tenant DBA	Tenant DBA will be responsible for managing security and access control of their organization databases.
Database sharding	Tenant DBA	MDaaS will support automated periodic sharding and manual sharding. Tenant DBA can make use of this feature for improving performance and mobile location of its customers.

3.10.2 MDaaS Infrastructure Administration

MDaaS infrastructure administration involves performing a set of tasks such as management of RMDIC storage management and provisioning, RMDIC and BDC compute node provisioning and load balancing, etc. We will examine below these tasks under BDC and MDIC administration respectively.

3.10.2.1 BDC Administration

DBA Task	Description
BDC VM Provisioning	Provision new VMs Setting up policies for allocation / stopping VMs based on load Periodic health check of VMs and analyze fault tolerance data from syslogs and any collected crashes.
Image upgrades	Management of software upgrade of BDC VMs and host OS / firmware
BDC Backend storage configuration	Add storage devices for backend databases (customer data) Backend database tuning
Manage resiliency	Setting up backend database backup and restore policies

3.10.2.2 MDIC Administration

DBA Task	Description
RMDIC & CMDIC VM Provisioning	Provision new VMs (compute) Setting up policies for allocation / stopping VMs based on load
Image upgrades	Management of software upgrade of BDC VMs and host OS / firmware
Cache Management	Manage data cache, cache density, depth and tuning
RMDIC provisioning	Add new RMDIC nodes to the system

4 System Implementation

This chapter enumerates a high-level plan for implementation of the MDaaS project. The design illustrated in the earlier chapter will be followed meticulously as much as possible during implementation. We have identified the key modules for implementation in Section 4.1, followed by the target environment that will be used. A high-level test summary for the implementation is available in Chapter 5.

4.1 Implementation Summary – Application Examples

4.1.1 Testbed Schema Organization

Figure 4.1.1.1 shows a sample schema that demonstrates how multi-tenancy implementation was tested. We have four tenants who created tables in *RealEstateDB* schema with the following privacy requirements for their tables:

- REALTOR-INC created a *PropertyInfoTable* with Shared-Table multi-tenancy (Type-3)
- SANTA-CLARA-COUNTY-TAX-GOV created a *PropertyTaxTable* with Shared-table Multi-tenancy (Type-3)
- COLDWELL-BANKER-INC created a *FinanceTable* with Separate-Schema Multi-tenancy (Type-1)
- REAL_AGENCY_INC created a *AgentList* table with Shared-Schema (Type-2) Multi-tenancy.

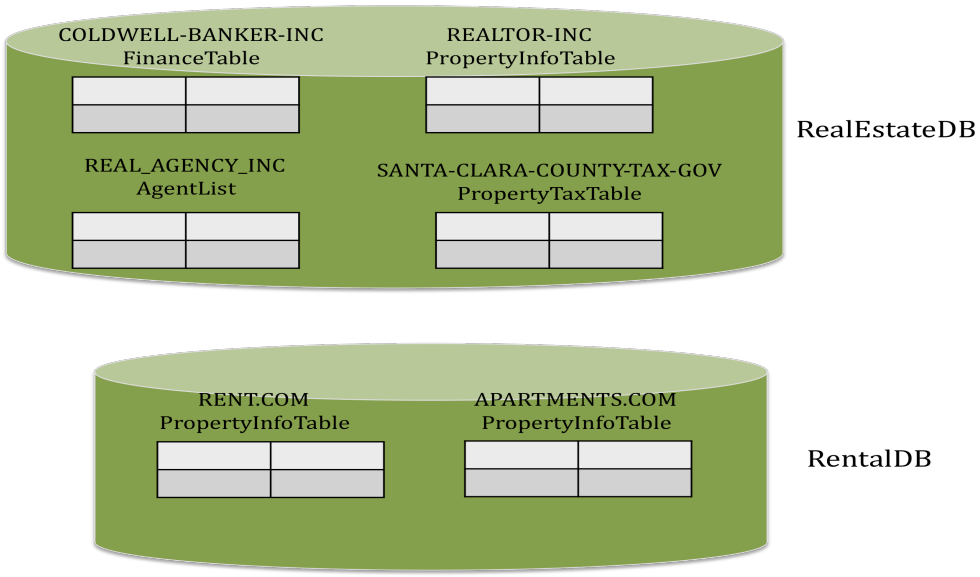


Figure 4.1.1.1: Original View of tenant Schema

Similarly, two tables were created by RENT.COM and APARTMENTS.COM with same name, viz. *PropertyInfoTable,* in RentalDB. We now see in Figure 4.1.1.2 how MDaaS multi-tenancy manager provisioned these tables. MDaaS maintains two metadata tables,

viz. TENANT_TABLE and COLUMN_TABLE, wherein mapping between the tenants and their desired multi-tenancy levels are maintained. This information is stored in a MDaaS-Metadata schema, which is private to MDaaS. MDaaS also creates three separate schema, one each for the three types of multi-tenancy, each of which contains a DATA_TABLE. The DATA_TABLE is the single storehouse of all tables created by the tenant.

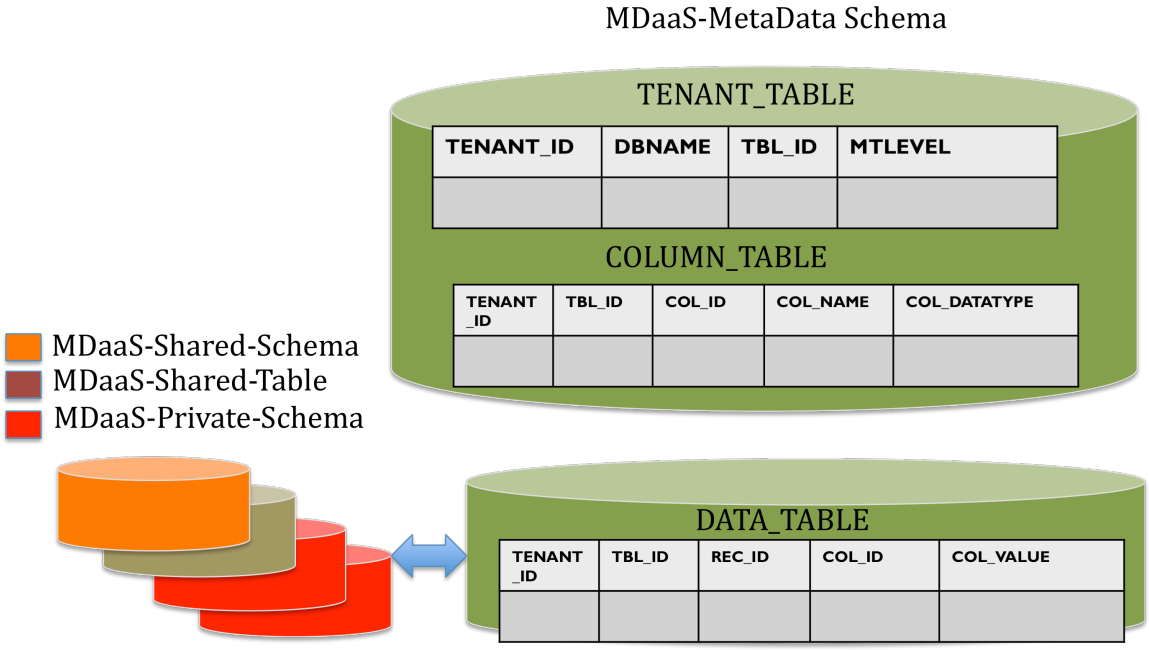


Figure 4.1.1.2: MDaaS View of tenant Schema

4.1.2 Mobile App Overview

A summary of apps that were created for testing is shown in the tables below. The customer apps are from the different tenants, which are primarily for data access.

Customer Apps

App Name	Vendor	Purpose
Financial	COLDWELL-BANKER-CORP	Type-1 (Private-Schema) MT
RealAgents	REAL_AGENCY_INC	Type-2 (Shared-Schema) MT
Realtor	REALTOR-INC	Type-3 (Shared-Table) MT
Taxman	SANTA-CLARA-COUNTY-GOV	Type-3 (Shared-Table) MT

The system apps are the ones that are created by MDaaS and distributed for administration purposes and for implementing MCDK.

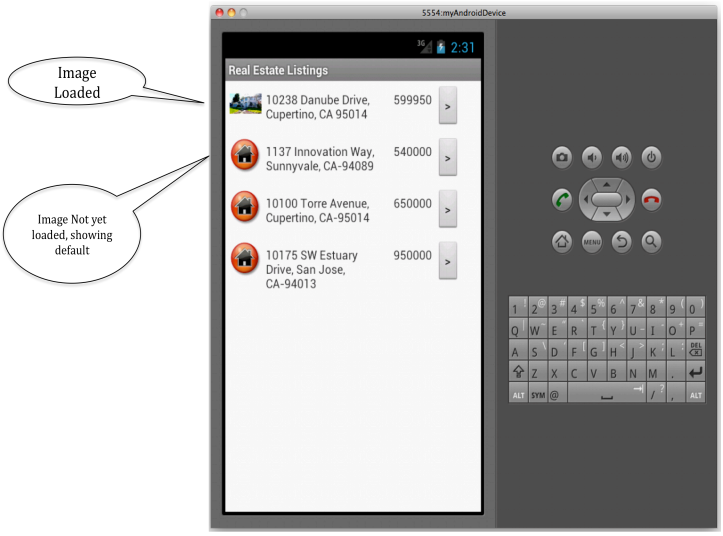
System Apps

App Name	Vendor	Purpose
MDaaSAdmin	MDaaS	DBA
MDaaSMobileDAL	MDaaS	Mobile Client Development Kit (MCDK)

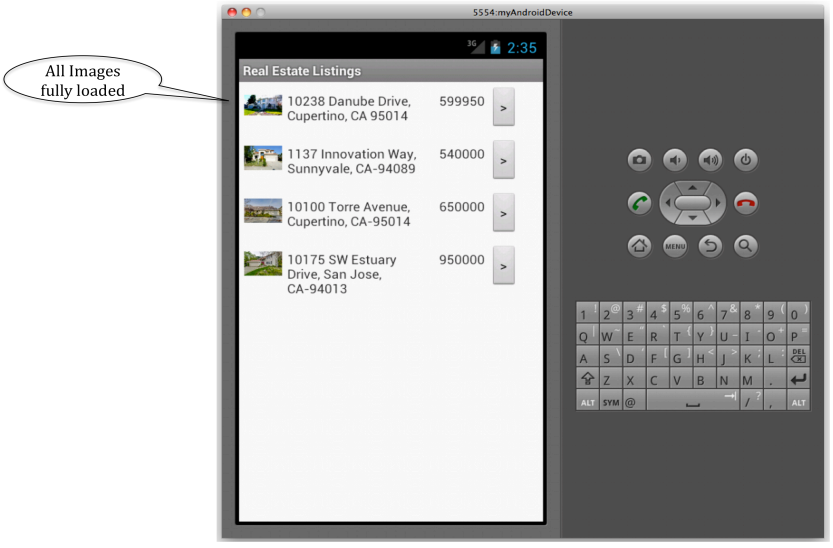
4.1.3 Example usage for User Accesses

We have shown in the figures below with embedded comments to demonstrate how accesses are made from the mobile app to MDaaS. Figure 4.1.3.1 illustrates that data responses are displayed first, followed by quick refresh of the screen once images get downloaded.

SCREEN 1 of 3: Android UI – Data loaded, images loading



SCREEN 2 of 3: Android UI – All Images and Data Loaded



SCREEN 3 of 3: Android UI – Using downloaded image and data in UI (other screens)

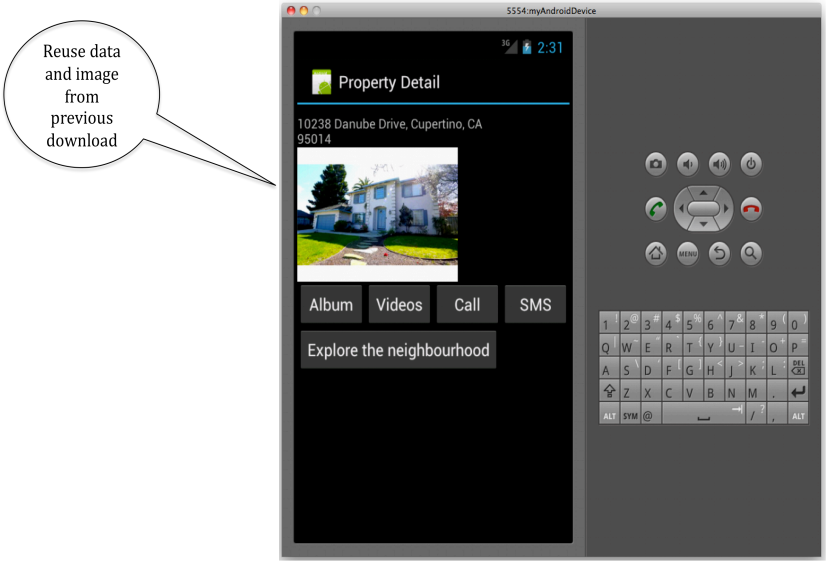
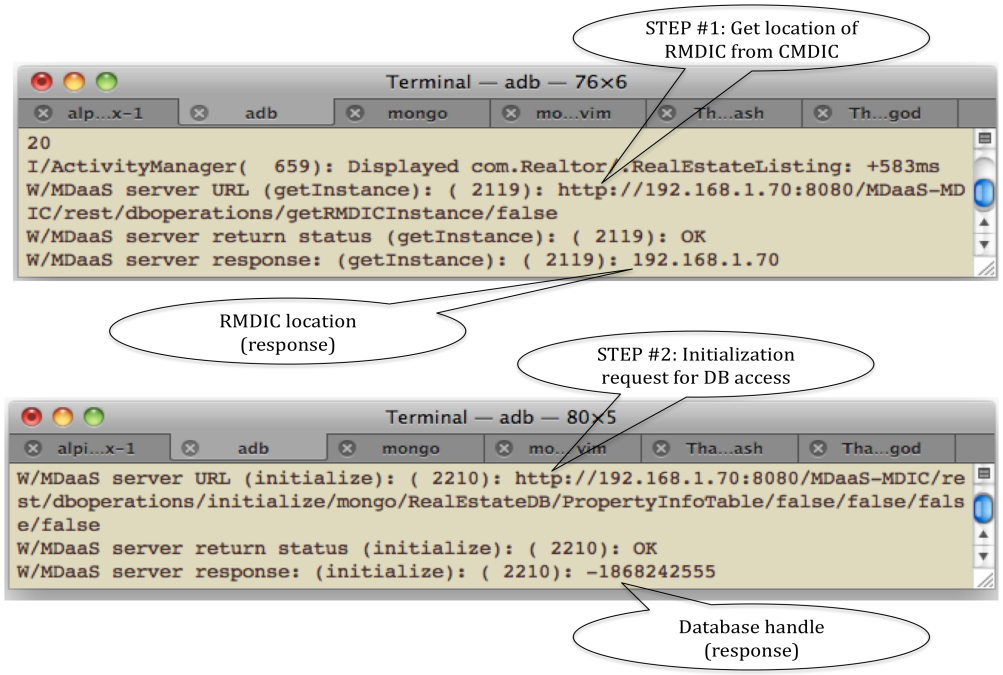


Figure 4.1.3.1: Data load & Image Loads.

We now demonstrate the actual sequence of events that take place as apps load the data and images. Figure 4.1.3.2 illustrations are examples from Android logs when communication between mobile device and MDaaS are in progress.



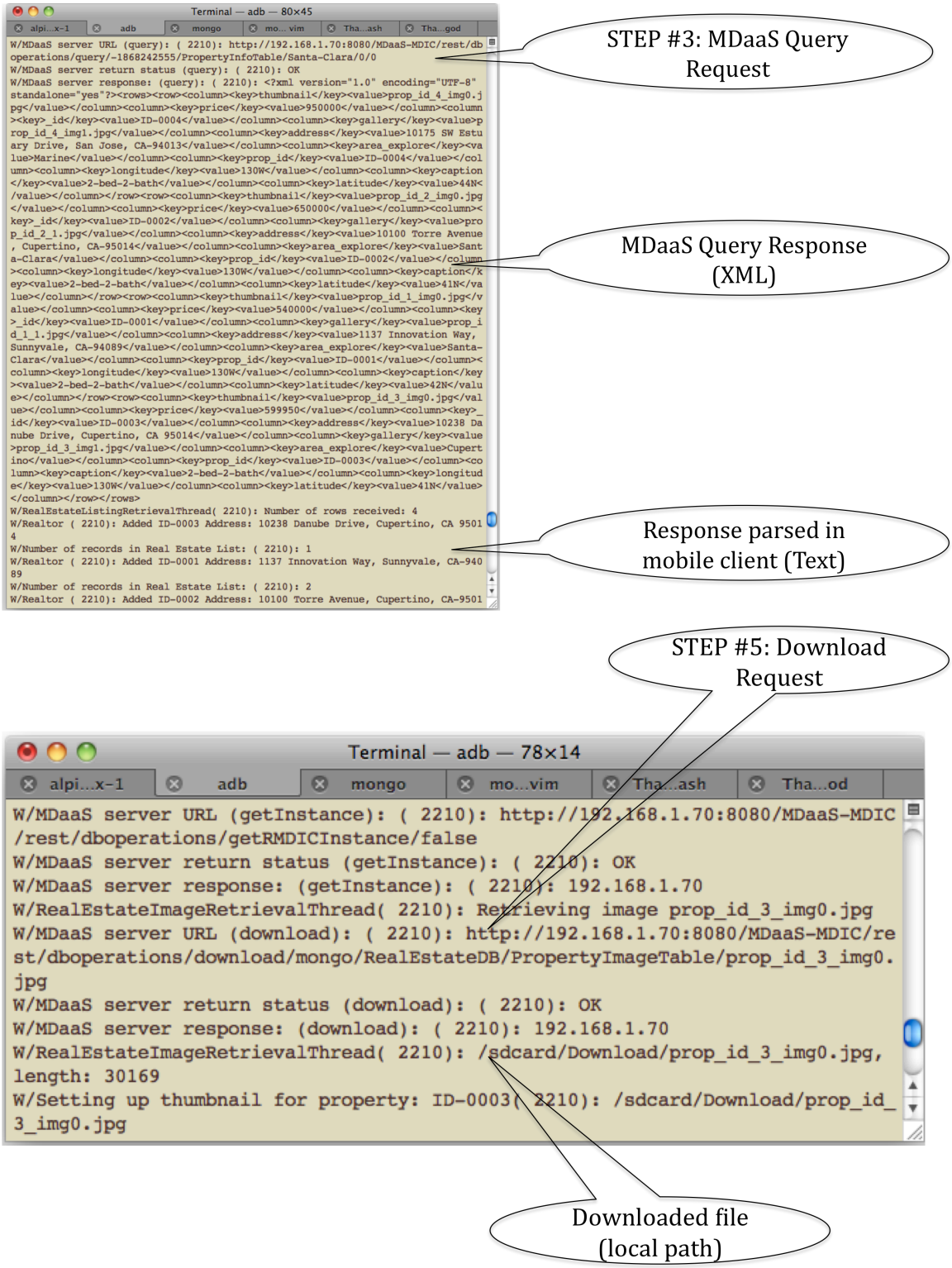


Figure 4.1.3.2: Mobile device to MDaaS communication examples

4.1.4 Example usage for Master DBA

We now illustrate two methods of administration of MDaaS. We first discuss the MDaaSAdminApp, followed by a discussion on the CLI tools for administration.

4.1.4.1 MDaaSAdminApp

Figure 4.1.4.1.1 displays screen shots of MDaaSAdminApp for displaying all tenants. We will subsequently add another tenant (CRAIGLIST) to the database through the app (Figure 4.1.4.1.2), and will show the listing again in Figure 4.1.4.1.3.

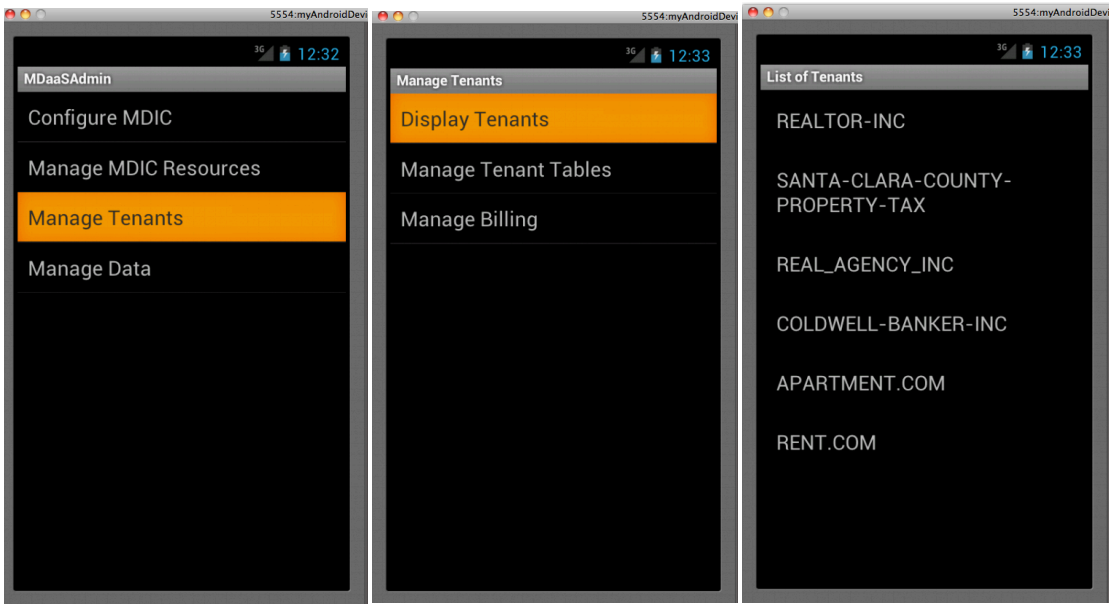


Figure 4.1.4.1.1: Displaying all tenants (before add)

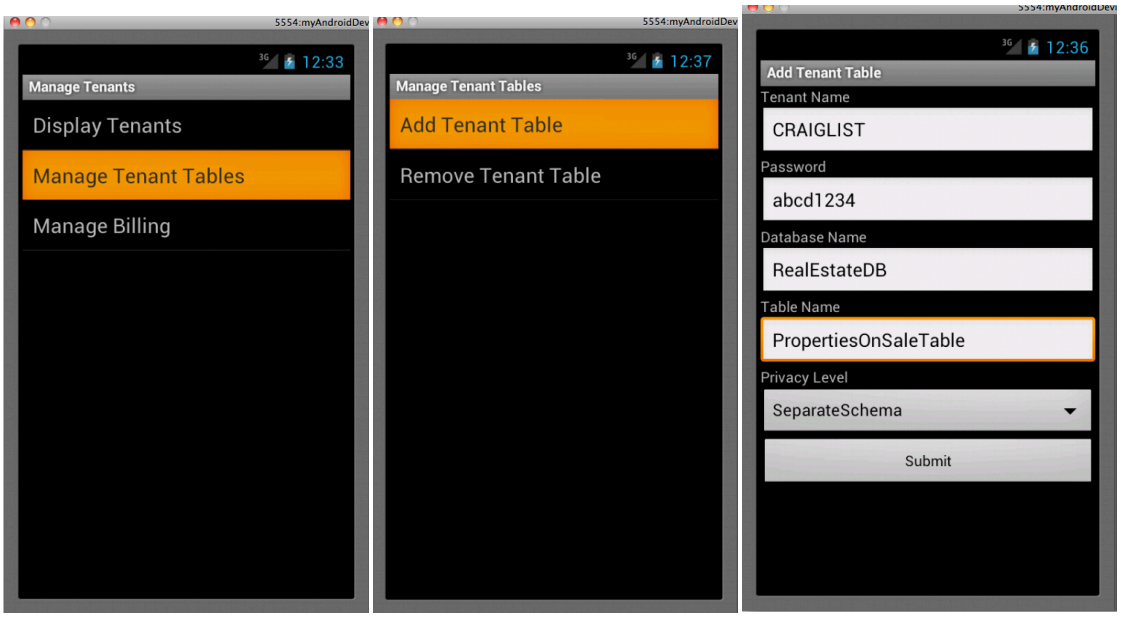


Figure 4.1.4.1.2: Adding a new tenant

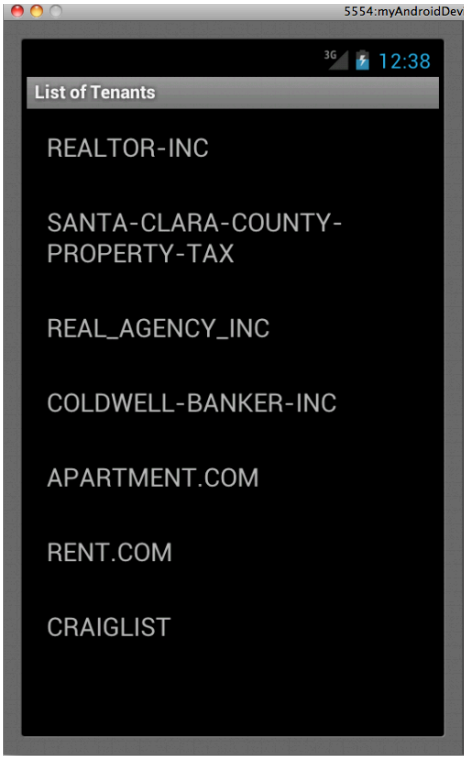


Figure 4.1.4.1.3: Displaying all tenants (after add)

We now illustrate in Figure 4.1.4.1.4 how to view format and tenancy of a particular table. This can be done by selecting a specified tenant in the tenant list and then selecting the desired schema/table. Similarly other CRUD operations on the schema and table are possible from the app, which are not shown here for brevity.

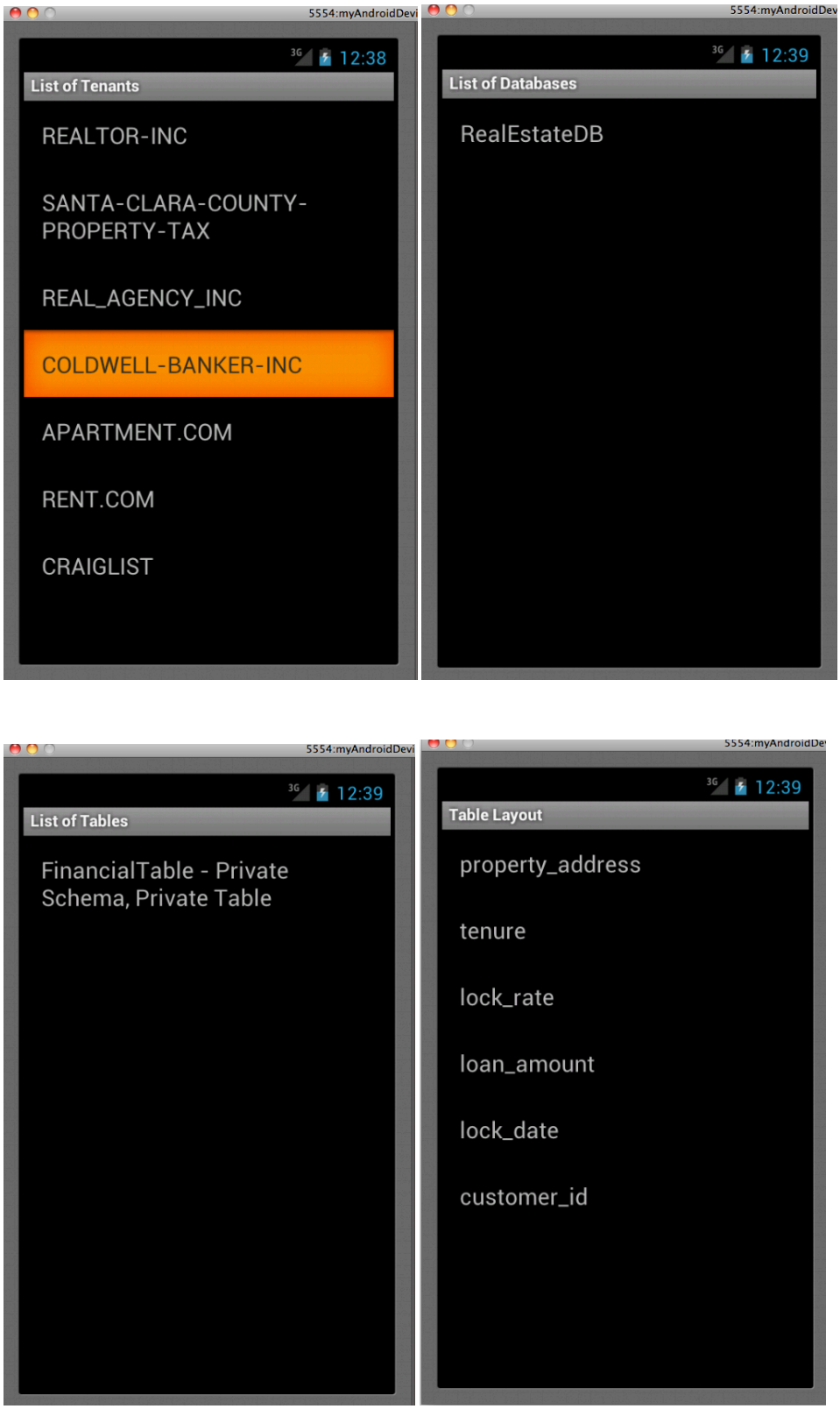


Figure 4.1.4.1.4: Displaying format & tenancy of a specified table/schema

4.1.4.2 MDaaS Administration CLI

For administering MDaaS, a set of CLI tools were developed that can perform CRUD operations on the databases, and tables. In first phase, we've used XML inputs to generate multi-tenant tables.

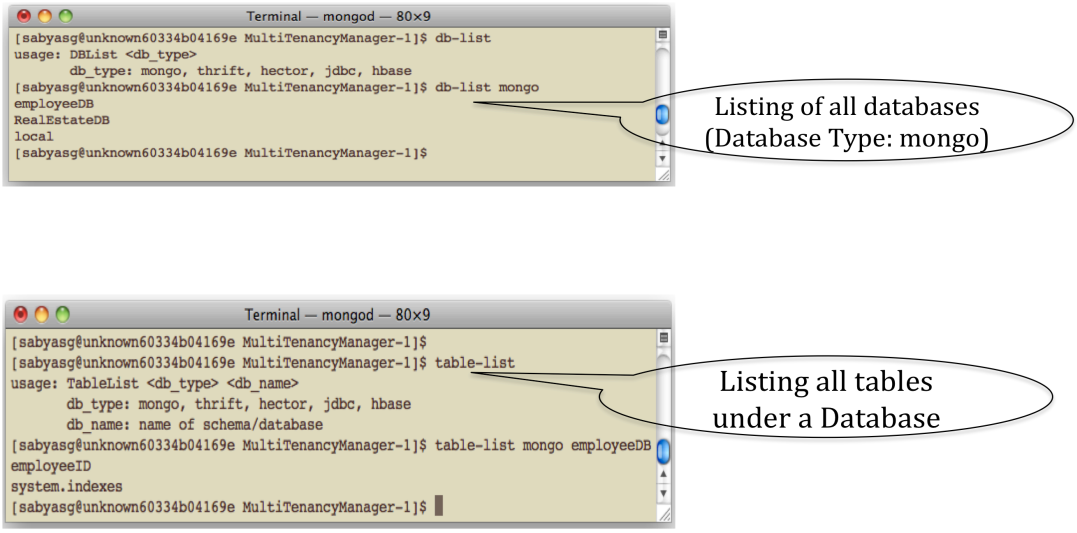


Figure 4.1.4.2.1: Listing all databases & tables

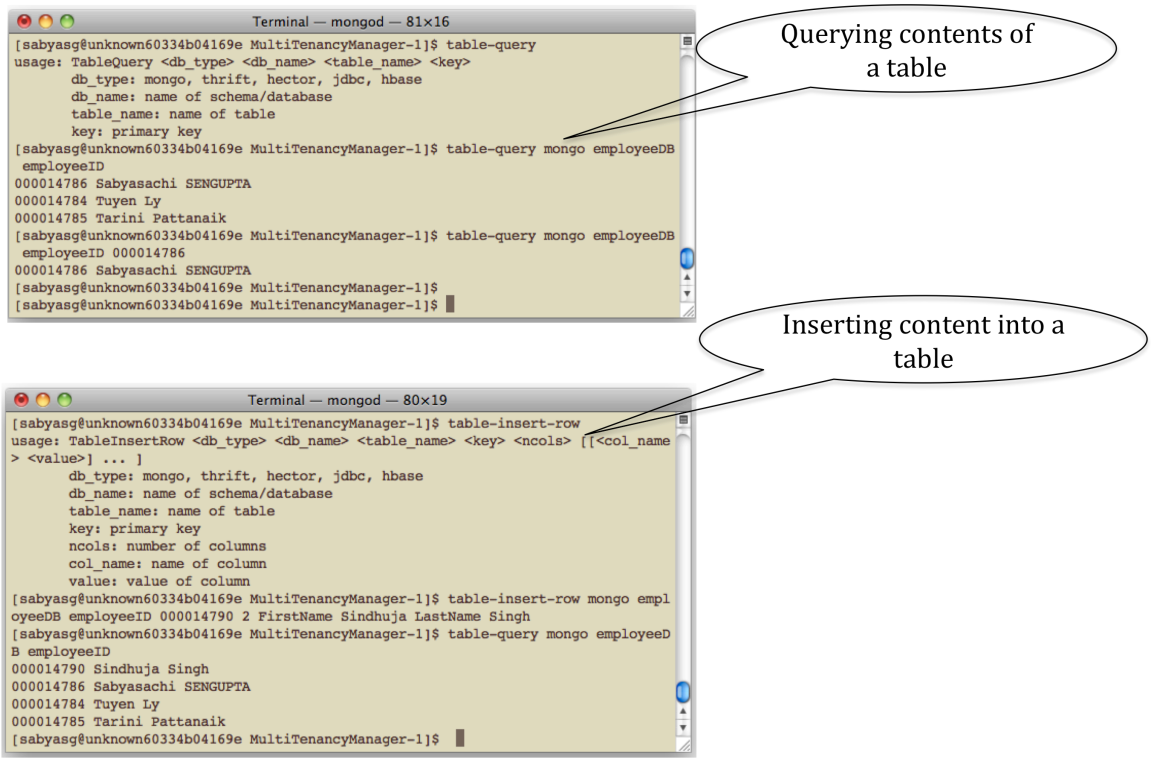


Figure 4.1.4.2.2: Querying and inserting tables

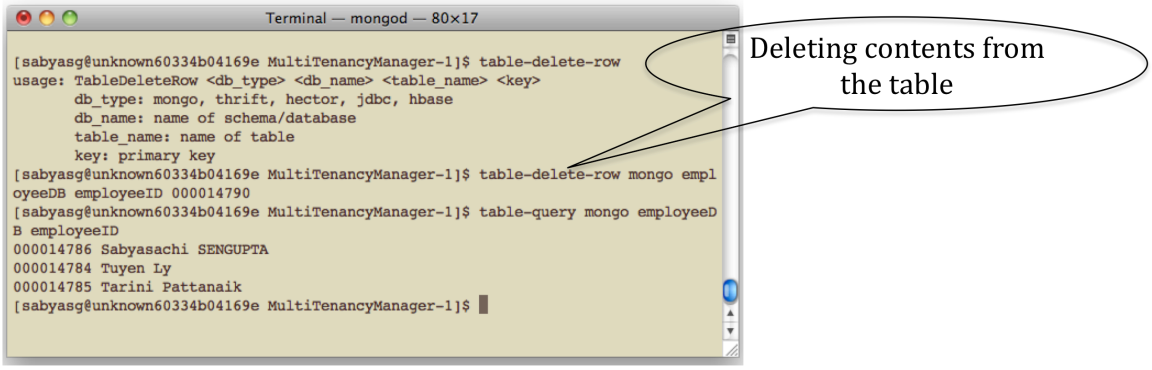


Figure 4.1.4.2.3: Deleting tables

4.2 Develop or Adopt Decision

Category	Item	Selection	Remarks
Software	Web Server	Tomcat	For UI and connection with mobile clients
	Web Application Environment	Apache Struts	For database accesses from Java front end
	Toolkits	JQuery	For UI programming
	Database	MySQL, Cassandra, MongoDB	For backend databases
	Load Balancer	Yahoo! Zookeeper	For load balancing

4.3 Implementation Process

The *Waterfall* model will be adopted for implementation. This means that emphasis will be laid on a particular module, which will be coded and unit tested first prior to moving to the next module. This will help in maintaining coherency in implementation.

4.4 Tools and Environments

Category	Item	Selection	Remarks
Infrastructure	Cloud Provider	Openstack	Cloud platform
	Cloud Storage	Swiftstack and Openstack block storage	Cloud storage
	Volume Manager	Linux LVM and RAID	Caching in RMDIC
Platform	Application language	Java	Development language
	Mobile End Client	Android	For mobile client applications
Tools	Version control	Subversion	
	Build	Ant	
	IDE	Eclipse and Android emulator	

4.5 Standards

For Java implementation, industry standard best practices will be used so that the code is easily maintainable and extensible in future. Coding standards improve readability and help engineering to maintain it easily. Code will be developed in the form of packages which should individually be unit tested. The coding guidelines and conventions presented by Oracle Java Coding Guidelines [59] will be followed.

4.6 Source Code Structure

In this section, we illustrate the source code organization of MDaaS solution. As seen in Figure 4.6.1, the source code will be laid out in a hierarchical fashion under key respective functional components, viz. the MDaaS Client components, the MDaaS server side components and the User Interface.

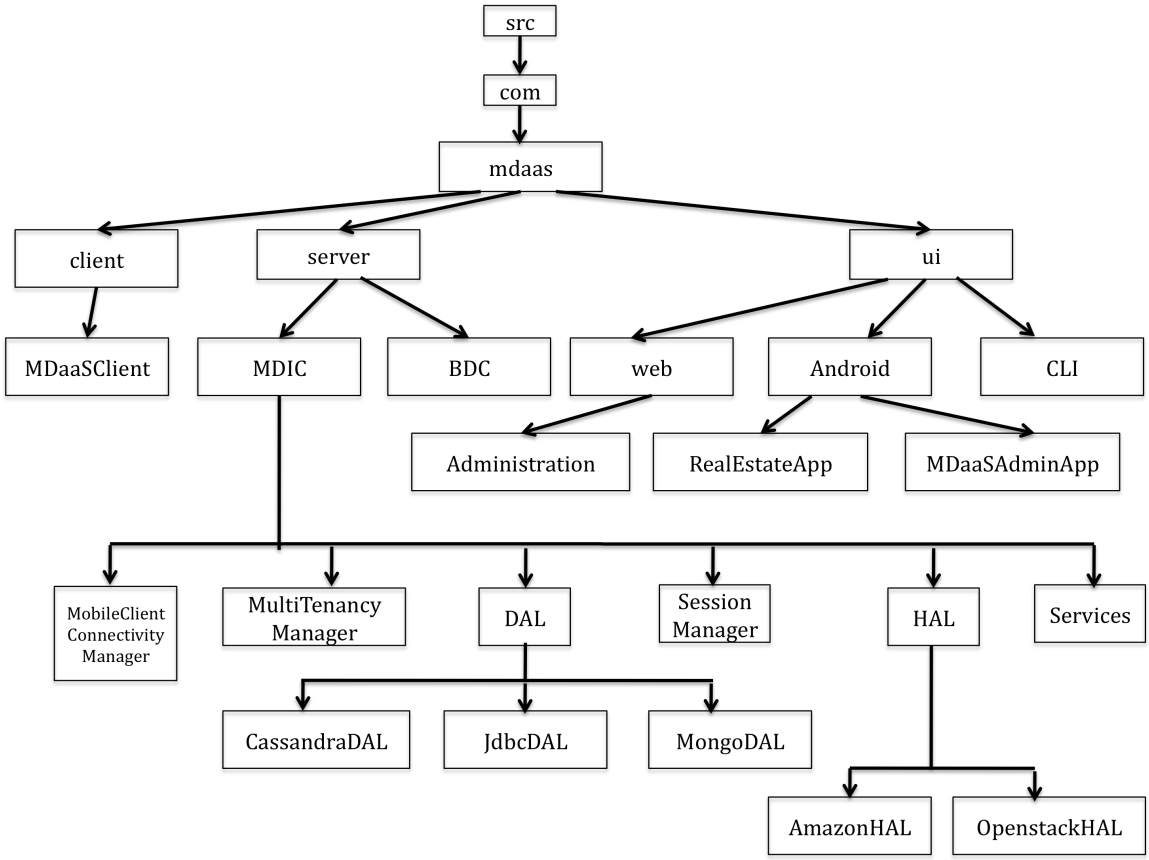


Figure 4.6.1: Source Code Layout

5 System Testing

5.1 Performance Analysis

Typical response time for queries from MDaaS depend on the following factors:

- Number of hops a query requires to be serviced:
  - Whether data present in Mobile DB (SQLite) – this is the fastest
  - Whether data present in user's nearest RMDIC Cache
  - Whether BDC access is needed? If BDC access is needed, all caches are updated following the access so that the subsequent accesses can be serviced without accessing the BDC.
- Number of fetches in a round trip for a query: If a query requires more fetches, total response time will be more.
- Processing power of the RMDIC: This depends on the following factors,
  - CPU/Memory Power of the VMs
  - Automatic provisioning of VMs on more load (elasticity)
- The size of the Database:
  - Larger the table, the more time taken by the BDC to respond
- Bandwidth
  - Connection speed depends on the ISP
  - QoS

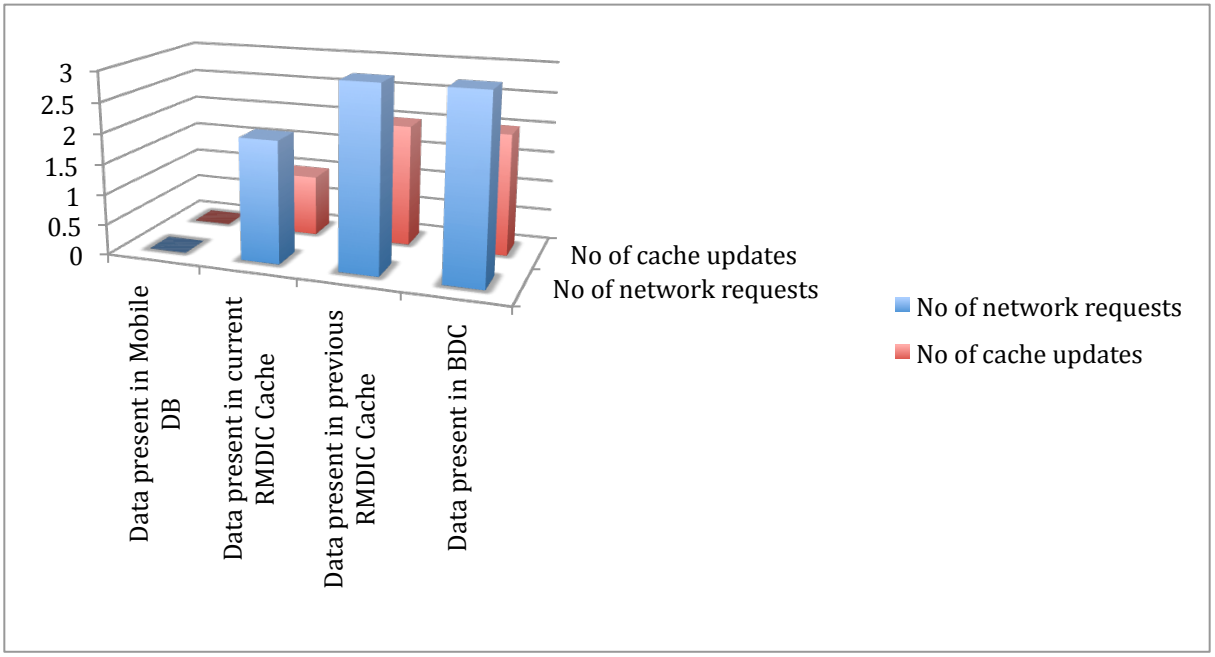


Figure 5.1.1: Network Requests and Cache Update Metrics for MDaaS

5.2 Test Infrastructure setup

This section provides an overview how the test infrastructure should be setup while constructing the code for MDaaS solution. In CMPE-295B timeframe, however, we didn't have the hardware that is required. Hence the prototyping was done in a simulated environment using local systems.

5.2.1 Mobile Data Infrastructure Cloud

An Openstack compute infrastructure should be setup [A.1], which will host different MDIC servers.

5.2.1.1 Central Mobile Data Infrastructure Cloud

The CMDIC should be deployed on two virtual machines in *Active-Active* pair so that it will be resilient. The following configuration is recommended for CMDIC VMs.

- *OS:* Oracle Enterprise Linux 32 bit
- *RAM*: 4G
- *CPU*: 2 (with SMP enabled)
- *Disk:* 16G boot drive (for Linux installation packages), and 32G data drive (for CMDIC databases).

In order to be able to support the above configuration, the host computer must have the following:

- *OS*: Oracle Enterprise Linux 64 bit
- *Hypervisor:* Oracle Virtual Box or QEMU
- *RAM*: 8G
- *CPU*: 4
- *Disk*: 128G (for base OS Linux)

5.2.1.2 Regional Mobile Data Infrastructure Cloud

At least one RMDIC server will be required for prototyping, which will have VMs that will run the actual MDaaS software. The VMs will be similar configuration as shown above for CMDIC. The configuration of RMDIC is also similar to that of CMDIC.

5.2.2 Backend Database Cloud

BDC functionality will be hosted on 4 VMs.

- *BDC Web Server:* This VM will host the BDC Administration Console and web interface
- *BDC Database Servers:* Two VMs will host the BDC databases in *Active-Active* mode so that it can be appropriately load balanced. As discussed in Chapter 3, Zookeeper will be used to coordinate and load balance the databases.
- *BDC Zookeeper front-end load balancer leader:* This VM will host the load-balancing leader, which will forward requests to the right DB server.

The configuration of the BDC VMs is recommended:

- *OS:* Oracle Enterprise Linux 32 bit
- *RAM*: 1G
- *CPU*: 1

- *Disk:* 16G boot drive (for Linux installation packages), and 16G data drive (for BDC databases).

The configuration of BDC host server will have to be higher than that of RMDIC as it will host more number of VMs. The configuration of the BDC host server should be the following:

- *OS*: Oracle Enterprise Linux 64 bit
- *Hypervisor:* Oracle Virtual Box or QEMU
- *RAM*: 16G
- *CPU*: 4 (with SMP enabled)
- *Disk*: 256G (for base OS Linux)

5.2.3 Storage organization

Persistent storage will be attached to each VM using LVM and RAID [A.2, A.3 and A.4].

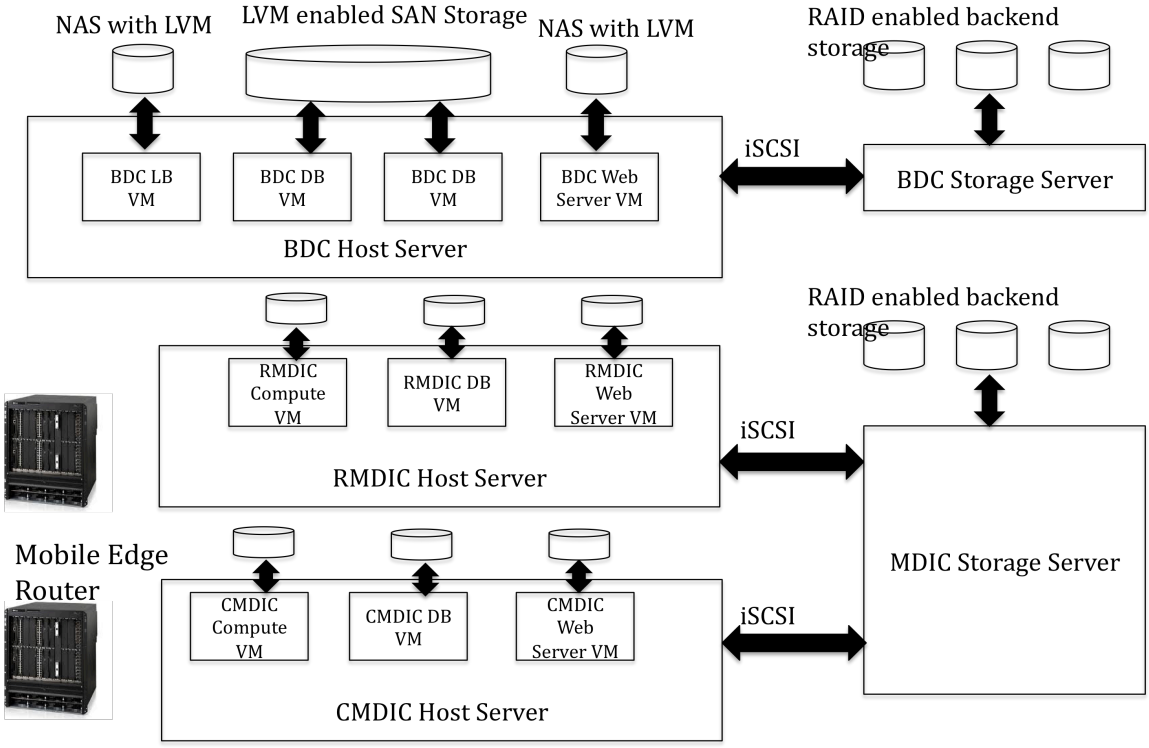


Figure 5.2.1: MDaaS Implementation Testbed

6 Conclusion and Future Work

The current work proposes a novel approach to efficiently manage and serve data for mobile computers. MDaaS design has many unique cloud features such as multi-tenancy, mobile database integration, and data level access abstraction along with automatic transaction failover with the ability to restart aborted sessions. Till date very few mobile cloud database solutions in the industry performs all these capabilities. We believe adopting the MDaaS solution will provide a major performance improvement for data access in the mobile communication industry. In the current timeframe, we have demonstrated implementation for some of the key aspects of the design of MDaaS, viz.

- *Database Abstraction Layer* that prevents technology lockin for customer apps
- Highly granular *Multi-tenancy* solution that enables setting different levels of privacy
- A mobile database solution that can sync up MDaaS upon data downloads
- A *Mobile Client Development Kit* that allows customer apps to have minimum intelligence about MDaaS connection and operational semantics, and helps in creating customer apps that implements application logic only without having to worry about database and its storage needs.
- Mobile apps and administration tools (CLI and mobile based) that demonstrate how to administer the system

We believe that the work can be transformed into a product by deploying it in a real time mobile network environment where connections tear down frequently, can be bursty along with considerations for handling user mobility. This requires having the capability to manage sessions and transfer state of the services along with cached data from one region to another seamlessly. As volume of data transferred and number of connections in a mobile environment can be extremely high, scalability is a major consideration. We have shown how the solution can scale by elastically provisioning Virtual Machines, which needs to be tested in real time environment. Thus load balancing of the services with Zookeeper needs to be implemented. Finally, more administrative services like billing, resource management and allocation by monitoring usage needs to be implemented.

References

[1] Craig D Weissman and Steve Borowski "*The Design of the Force.com Multitenant Internet Application Development Platform."* Salesforce.com. SIGMOD '09, Providence, Rhode Island, USA, ACM 978-1-60558-551-2/09/06, June 29-July 02, 2009.

[2] Hassan Artail, Haidar Safa, Rana ELZinnar, and Hicham Hamze, *"A Distributed Database Framework from Mobile Databases in MANETs,"* Proceedings of Third IEEE International Conference in Wireless and Mobile Computing, Networking and Communications, WiMOB, 2007.

[3] Eun-Hee Hyun and Sung-Hee Kim. "*Real-time Mobile Data Management Using MMDB"*. Proceedings of Second International Workshop on Real-Time Computing Systems and Applications, 1995.

[4] Dik Lun Lee, Jianliang Xu, Baihua Zheng, and Wang-Chien Lee. "*Data management in location-dependent information services"*. Proceedings of 20th International Conference on IEEE Pervasive Computing and Data Engineering, 2004.

[5] Patrick Stuedi, Iqbal Mohomed, and Doug Terry, *WhereStore: Location-based Data Storage for Mobile Devices Interacting with the Cloud,* Technical Report, Microsoft Corporation, 2010.

[6] N Marsit, A Hameurlain, Z Mammeri, F Morvan, "*Query processing in mobile environments: a survey and open problems*," First International Conference on Distributed Frameworks for Multimedia Applications, DFMA, 2005.

[7] Zhibin Zhou and Dijiang Huang, "*Efficient and Secure Data Storage Operations for Mobile Cloud Computing"*, Arizona State University, Retrieved from: http://eprint.iacr.org/2011/185.pdf

[8] Li Ye-bai, Zhang Bin, and Wang Hai-bin, *"Study and Design on Data Management Model of SQL Server CE for Mobile Applications",* Proceedings of International Conference on e-Education, e-Business, e-Management, and e-Learning. IEEE Computer Society, 2010.

[9] Chris Kanaracus, "*Database gurus tackle cloud challenges"*. Infoworld. Retrieved from: http://www.infoworld.com/d/cloud-computing/database-gurus-tackle-cloud-challenges-852, Jan 28, 2011.

[10] Wu Xinhua, Li Liu, *"Location Dependent Continuous Queries Processing Model Based on Mobile Agents"*. Proceedings of Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science, 2010.

[11] Jung Hun Kang, Wonil Choi, Myong-Soon Park, "*Efficient Load Balancing Method For Mobile Applicable iSCSI based Remote Storage Service",* Proceedings of the Fourth International Conference on Software Engineering Research , Management and Applications. IEEE Computer Society, 2010.

[12] Darrin Chan and John F Roddick, "*Summarization of Mobile Databases*". Australian Computer Society Inc., School of Informatics and Engineering, Flinders University of South Australia, 2005

[13] Anne Marie Amja, Abdel Obaid, and Normand Seguin, "*A Distributed Mobile Database Architecture,*" Proceedings of Services Computing Conference (APSCC), IEEE Asia-Pacific,2011.

[14] Guy Bernard, Jalel Ben-Othman, Luc Buoganim, Gerome Canals, Sophie Chabridon, et al. "*Mobile Databases: A Selection of Open Issues and Research Directions"*. Newsletter ACM SIGMOD Record, 2004.

[15] V. Mateljan, D. Cisic, and D. Ogrizovic, *"Cloud Database-as-a-Service (DaaS) - ROI,"* Proceedings of the 33rd International Convention, MIPRO, 24-28 May 2010.

[16] Shalini Ramanathan, Savita Goel, and Subramanyan Alagumalai, *"Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable,"* Proceedings of International Conference on Recent Trends in Information Systems (ReTIS), 2011.

[17] Shyam Kotecha, Minal Bhise, and Sanjay Chaudhary, *"Query translation for cloud databases,"* Nirma University International Conference on Engineering (NUiCONE), 2011.

[18] Han Jing, Son Meina, and Song Junde, *"A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing,"* Proceedings of IEEE/ACIS 10th International Conference on Computer and Information Science (ICIS), 2011.

[19] Minpeng Zhu and T Risch, *"Querying combined cloud-based and relational databases,"* Proceedings of International Conference on Cloud and Service Computing (CSC), 2011.

[20] H. Hacigumu. B. Iyer and S. Mehrotra, "*Providing database as a service*," Proceedings of 18th International Conference on Data Engineering, 2002 .

[21] Dejan Kovachev, Yiwei Chao, and Ralf Klamma, *"Mobile Cloud Computing: A Comparison of Application Models",* Information Systems and Database Technologies, RWTH Aachen University, Germany, 2011.

[22] Jerry Zeyu Gao, Simon Shim, Hsing Mei, Xiao Su, *"Engineering Wireless-Based Software Systems and Applications",* Artech House, 2006.

[23] Robin Bloor, "*What is a Cloud Database? The suitability of Algebraix Data's Technology to Cloud Computing*", The Bloor Group, 2011.

[24] Mazedur Rahman, Jerry Gao, "*An Energy-Saving Service System for Cloud Computing",* San Jose State University.

[25] Amreen Khan, Kamalkant Ahirwar, "*Mobile Cloud Computing as A Future of Mobile Multimedia Database"* International Journal of Computer Science and Communication, Vol-2, No. 1, January-June 2011.

[26] Eugene E. Marinelli, *"Hyrax: Cloud Computing on Mobile Devices using MapReduce*", School of Computer Science Carnegie Mellon University, September 2009.

[27] Bharath Cheluvaraju, Aravalli Srinivasa- Ramachandra Kousik and Shrisha Rao, "*Anticipatory Retrieval and Caching of Data for Mobile Devices in Variable Bandwidth Environments",* 5th Annual IEEE International Systems Conference (IEEE SysCon 2011), Montreal, Canada, April 2011

[28] S Ramanathan, "*Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable",* International Conference on Recent Trends in Information Systems (ReTIS), 2011

[29] S Ganesh, Mathan Vijayalakshmi, A Kannan, "*Intelligent Agent Based Approach for Transaction Processing in Mobile Database Systems"*, The International Arab Journal of Information Technology, Vol. 4, No. 2, April 2007

[30] J H Abawajy, M Mat-deris, "*Supporting Disconnected Operations in Mobile Computing*," Computer Systems and Applications, 2006 doi: 10.1109/AICCSA.2006.205197

[31] Lu Qi, M Satyanarayanan, "*Improving data consistency in mobile computing using isolation-only transactions*," Hot Topics in Operating Systems, 1995, doi: 10.1109/HOTOS.1995.513467

[32] Li Minjie, Liqiang Wang, Ying Hao, "*A mobile agents based framework in wireless network*" Information Management and Engineering (ICIME), 2010 doi: 10.1109/ICIME.2010.5477509

[33] M Tarafdar, M S Haghjoo, "*Location privacy in processing location dependent queries in mobile database systems*," Telecommunications (IST), 2010 doi: 10.1109 /ISTEL.2010.5734021

[34] D Barbara, "*Mobile computing and databases-a survey*," Knowledge and Data Engineering, IEEE Transactions on, 1999 doi: 10.1109 /69.755619

[35] Z Wang, S K Das, H Che, M Kumar, "*A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environment*", IEEE Transactions in Computer Science (Parallel and Distributed Systems). Vol 15, No 11, November 2008.

[36] D A Adjeroh, K C Nwosu, "*Multimedia database management-requirements and issues,*" MultiMedia, IEEE, vol.4, no.3, pp.24-33, Jul-Sep 1997 doi: 10.1109/93.621580

[37] Y Liu, Y Wang, Jin Yi , "*Research on the improvement of MongoDB Auto-Sharding in cloud environment*," Computer Science & Education (ICCSE), 2012 7th International Conference on , vol., no., pp.851-854, 14-17 July 2012 doi: 10.1109/ICCSE.2012.6295203

[38] [Unattributed], *Scaling MySQL to new heights,* ScaleDB, Retrieved from: http://www.scaledb.com/architecture.html

[39] Wilson A, *"Distributed Transactions and Two-Phase Commit"* Retrieved from: http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/3732d690-0201-0010-a993-b92aab79701f?overridelayout=true

[40] Abdul-Mehdi, Z.; Bin Mamat, A.; Ibrahim, H.; Deris, M.M.; , "*A model for transaction management in mobile databases*," Potentials, IEEE , vol.29, no.3, pp.32-39, May-June 2010 doi: 10.1109/MPOT.2010.936929 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5458474&isnumber=5458448

[41] Munakata, S.; Saito, K.; Hiji, M.; "*A new session management method for efficient mobile access to web applications*," Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on, vol., no., pp.497-502, 8-11 July 2008 doi: 10.1109/CIT.2008.4594725 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4594725&isnumber=4594630

[42] Frederick Chong, Gianpaolo Carraro, and Roger Wolter, "*Multi-tenant Data Architecture – MSDN Architecture Center".* June/2006. Microsoft Corporation, Retrieved from: http://msdn.microsoft.com/en-us/library/aa479086.aspx

[43] Vijay Kumar, Nitin Prabhu, and Panos K Chrysanthis; "*HDC – Hot Data Caching in Mobile Systems";* Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on , vol., no., pp. 37, 2005 doi: 10.1109/AICCSA.2005.1387034 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1387034&isnumber=30191

[44] Yimeng Liu; Yizhi Wang; Yi Jin;, "*Research on the improvement of MongoDB Auto-Sharding in cloud environment*," Computer Science & Education (ICCSE), 2012 7th International Conference on , vol., no., pp.851-854, 14-17 July 2012 doi: 10.1109/ICCSE.2012.6295203 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6295203&isnumber=6295013

[45] Tamishetty, R.; Lek Heng Ngoh; Pung Hung Keng; , "*Query-based Wireless Sensor Storage Management for Real-time Applications*," Industrial Informatics, 2006 IEEE International Conference on , vol., no., pp.166-170, 16-18 Aug. 2006 doi: 10.1109/INDIN.2006.275754 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4053381&isnumber=4053336

[46] O'Connor, M.; Andrieu, V.; Roantree, M.; , "*Query Management in a Sensor Environment*," Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on , vol., no., pp.835-840, 8-10 Dec. 2008 doi: 10.1109/ICPADS.2008.25 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4724404&isnumber=4724287

[47] Wen-Chih Peng; Ming-Syan Chen; , "*Query processing in a mobile computing environment: exploiting the features of asymmetry*," Knowledge and Data Engineering, IEEE Transactions on , vol.17, no.7, pp. 982- 996, July 2005 doi: 10.1109/TKDE.2005.115 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1432706&isnumber=30889

[48] Shiow-Yang Wu, and Kun-Ta Wu; "*Dynamic Data Management for Location Based Services in Mobile Environment",* Database Engineering and Applications Symposium, 2003. Proceedings. Seventh International , vol., no., pp. 180- 189, 16-18 July 2003 doi: 10.1109/IDEAS.2003.1214925 Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.6204&rep=rep1&type=pdf

[49] Adams, K.P.; Gracanin, D.; Hinchey, M.G.;, "Increasing resiliency through priority scheduling of asynchronous data replication," Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on , vol.1, no., pp. 356- 362 Vol. 1, 20-22 July 2005 doi: 10.1109/ICPADS.2005.171 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1531150&isnumber=32668

[50] [Unattributed], "*Enterprise Mobile Cloud Computing – Cloud Services, Mobile devices, and IT Supply Chain Analysis*". ABI Research, Retrieved from: http://www.abiresearch.com/research/product/1004607-enterprise-mobile-cloud-computing/

[51] [Unattributed]. "*US Federal Cloud Computing Forecast – 2013-2018"*. Market Research Media. Retrieved from: http://www.marketresearchmedia.com/?p=145

[52] Saurabh Bagchi. *"Three Significant Trends in Mobile Security: Threats and Solutions"*. Purdue University. Retrieved from: https://engineering.purdue.edu/dcsl/presentations/2012/cerias_panel_mobilesecurity_040312.pdf

[53] [Unattributed]. "*Mobile Cloud Computing: devices, trends, issues and enabling technologies"*. IBM Research. Retrieved from: http://www.ibm.com/developerworks/cloud/library/cl-mobilecloudcomputing/

[54] [Unattributed]. "*Five Major Trends in Cloud Computing"*. CloudTimes. Retrieved from: http://cloudtimes.org/2011/06/05/5-major-trends-in-mobile-cloud-computing

[55] Barunch Awerbuch, and Herb Rubens; "*Wireless Networks Research Overview";* Retrieved from: http://www.cs.jhu.edu/~dholmer/600.64/Flecture1-modified.ppt&ei=yuKXUNqQA6K82wXNj4GAAw&usg=AFQjCNGFeOvrnrBLGQlsLyX-vgRzkMW09g&sig2=c7C4U8YbtMFssjFI02FL1A

[56] Jim Florick, Jim Whiteaker, Alan C Amrod, Jake Woodhams; "*Wireless LAN Design Guide for High Density Client Environment in Higher Education"*; November 2011. Retrieved from: http://www.cisco.com/en/US/prod/collateral/wireless/ps5678/ps10981/design_guide_c07-693245.pdf

[57] Fei Shi; Mefford, C.; , "A New Indexing Method for Approximate Search in Text Databases," Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on , vol., no., pp.70-76, 21-23 Sept. 2005 doi: 10.1109/CIT.2005.23 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1562630&isnumber=33169

[58] [Unattributed]. "*Apache Hadoop Zookeeper – Administrative Guide"*. Retrieved from: http://zookeeper.apache.org/doc/r3.3.3/zookeeperAdmin.html

[59] [Unattributed]. "*Code Conventions for the Java Programming Language"*. Oracle Corporation. URL: http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

[60] [Unattributed]. "*CouchDB – Complete API Reference"*. CouchDB. Retrieved from: http://wiki.apache.org/couchdb/Complete_HTTP_API_Reference

[61] Lei Zhang, Mahesh Jain, Anand Sherkhane, *"Enhance A Multi-Tenant Cloud No SQL DB Modeling Tool"* (2012). CMPE-281, Cohort-3 Project Report, San Jose State University

[62] Raul F Chong. *"Designing a database for multi-tenancy in cloud"*. (2012). IBM Developer Works. Retrieved from: http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/index.html

[63] Frederick Chong, G Carraro, Roger Wolter (2006). *Multi-tenant Data Architecture.* Microsoft Research. Retrieved from: http://msdn.microsoft.com/en-us/library/aa479086.aspx

Appendices

A Procedures to setup testbed

A.1 Configuring Openstack Compute Cloud

1. Install VirtualBox.
2. Download an Ubuntu distribution (12.04) ISO, which will be used as the base virtual machine. Provision Ubuntu VM.
3. Clone *devstack* repository into the Ubuntu VM from its shell:
   ```
   # git clone git://github.com/openstack-dev/devstack.git
   ```
4. Start the installation for deploying the Openstack cloud from the Ubuntu shell:
   ```
   # cd devstack; ./stack.sh
   ```
5. Installation takes about ~20 minutes. Once installation is complete, it ends with the following lines on the console:
   ```
   Horizon is now available at http://10.0.2.15/
   Keystone is serving at http://10.0.2.15:5000/v2.0/
   Examples on using novaclient command line is in exercise.sh
   The default users are: admin and demo
   The password: sabyasg
   This is your host ip: 10.0.2.15
   stack.sh completed in 1279 seconds.
   ```

A.2 Configuring Logical Volume Manager (LVM)

- Setting up Physical Volumes
  ```
  # pvcreate <list_of_physical_devices>
  # pvdisplay [<physical_device>]
  ```
- Setting up Volume Group
  ```
  # vgcreate <name_of_vg> <name_of_physical_devices>
  # vgdispay [<vg_name>]
  ```
- Setting up Logical Volume
  ```
  # lvcreate --size <size_of_partition> --name /dev/vg00/lvol1
  # lvdisplay [<lv_name>]
  ```

A.3 Configuring Snapshots with Logical Volume Manager

- Create snapshot drive *snapdrive* with LVM (size 50M).
  ```
  # lvcreate -L 50M --snapshot --name snapshot /dev/vg00/lvol1
  ```
- Take requisite backups after mounting the filesystem
- Remove snapshot volume
  ```
  # lvremove /dev/vg00/snapshot
  ```

A.4 Configuring RAID

- Setting up RAID-1 on two SATA drives in Linux

```
# mdadm --create /dev/md0 --raid-devices=2 /dev/sda1 /dev/sdb1
```

- Checking status of the RAID drives

```
# mdadm --query --detail /dev/md0

# cat /proc/mdstat
```

- Setting up automatic health check monitor on RAID drives (every 60 seconds)

```
# mdadm --monitor /dev/md0 --delay 60 --daemonize --program
<path_to_health_checker_daemon>
```