CHAPTER THREE

SPECIFICATIONS OF THE REPLICATION STRATEGY

3.0 Overview

To cope with the research problem, a new replication strategy is proposed. The proposed replication strategy encompasses four components: replication architecture, updates propagation protocol, replication method, and updates ordering mechanism. The purpose of the replication architecture is to provide a solid infrastructure for distributing replicas among wide areas, improving data availability, and supporting large number of replicas in mobile environments by determining the required components that are involved in the replication process. The purpose of the propagation protocol is to transfer data updates between the components of the replication architecture in a manner that achieves the consistency of data and improves the availability of recent updates to all interested hosts. Updates ordering mechanism ensures a unified causal and total ordering for all updates occurred in the replication system. The replication method provides a mechanism for implementing the propagation protocol, updates ordering, and allocation of replicas to the different components of the replication architecture.

The new strategy is a hybrid of both pessimistic and optimistic replication approaches. The pessimistic approach is used for restricting updates of infrequently changed data to a single replica. The reason behind this restriction is that if the modifications of these data are allowed on several sites, it will influence data consistency by having multiple values for the same data item (such as multiple codes for the same disease or multiple codes for the same drug). On the other hand, the optimistic replication is used for allowing updates of frequently changed data to be performed on multiple replicas. The classification into frequently and infrequently changed data is specified

according to the semantic and usage of the data items during the design phase of the database.

In this chapter, a description is provided for the replication architecture and the replication method, while the description of updates propagation protocol and updates ordering mechanism is provided in chapter 4 and chapter 5, respectively.

3.1 System Model

This research considers a large-scale environment that consists of Fixed Hosts (FH), Mobile Hosts (MH), a replica manager on each host, and a replicated database on each host. A part of fixed hosts represent servers with more storage and processing capabilities than the rest. The replica manger is a software that manages and facilitates the application's accesses (read and update operations) to the replicated database. A replicated database is called mobile database when it is stored on a mobile host. The replicated database contains a set of objects stored on the set of hosts. The database is fully replicated on the servers, while it is partially replicated on both fixed and mobile hosts. Update can take place at any host. Update information is sent to the other hosts in a form of message. Each host has a unique address, called *Host- ID*. The information of hosts and their replicated data are stored on objects called *Hosts-Obj* and *Host-Replicated-Objects-Obj*, respectively, which are replicated in each server. The description of these objects is provided in appendix One. When a failure occurs in any host, the affected host is considered as disconnected. In this thesis, the terms *replica* and *host* will be used interchangeably because each host has a replica.

The different entities and concepts that are involved in the replication system are defined formally as follows.

Definition 3.1.1 An object O is the smallest unit of replication and it represents a tuple $O = <D, R, S>$, where $D = \{d_1, d_2, ..., d_n\}$ is a set of data items of the object O, $R = \{r_1, r_2, ..., r_m\}$ is a set of replicas of O, and S is the state of the object O.

Definition 3.1.2 The state S of an object O is a set consisting of states that identifies current values for each data item $d_i \in D$, i.e., $S = \{s_1, s_2, ..., s_n\}$.

Definition 3.1.3 A replica R is a copy of an object O that contains at least one data item of O and is stored in a different host. R is defined as a function as follows. For a set of

updates U that is performed on a set of objects $\bar{O}$, the function $R : U \times \bar{O} \rightarrow S$ identifies a new separate state $s_i \in S$ for an object $O \in \bar{O}$ as a result of performing update $u \in U$ on an object O in a different host.

Definition 3.1.4 For a replica R in a host H, Interested Data Items is a subset I of the set of all data items, which is required for H to perform its duties, i.e., $I \subseteq \{ \bigcup_{i=1}^{n} O_i \}$, where n is the number of objects in the system.

Definition 3.1.5 An update is a database access that is executed locally on each host and it affects the state of the accessed object by modifying it from the old state S to a new state S'.

Definition 3.1.6 A propagate is an operation that sends updates that occurred in a specific replica to the other replicas, which leads to affecting their states.

Definition 3.1.7 A read is a database access that is executed locally on each host and it does not affect the state of the accessed object.

Definition 3.1.8 A replicated data item $d_i \in D$ is consistent if and only if its values are identical and in the same order as the values of the similar data item in the replica that is stored in the master server, which exists in the fixed network.

Definition 3.1.9 A replica R is consistent if and only if each interested data item $d_i \in \{D \cap I\}$ is consistent.

Definition 3.1.10 A replica R in a mobile host is in Available-State for a data item $d_i \in D$ if and only if all updates that are performed on d_i in other replicas (either in fixed hosts or mobile hosts) are merged with the updates that are performed on d_i in R.

Definition 3.1.11 Consistent-Available State (CA State) for a replica R that is stored in a mobile host is the state in which:

1. R is consistent
2. R in Available-State for each interested data item d_i in R.

Every update propagated to the other hosts is marked using a data item called Propagation-Flag, which can be defined as follows.

Definition 3.1.12 Propagation-Flag is a database item that has one value, which is either T in case of the update is propagated to the other hosts or F in the case of the update is waiting to be propagated.

The number and placement of mobile replicas are not fixed, since they are changed according to the number of mobile hosts and their interested data items. At any time the user is expected to change its interesting data items, which leads to changing its replicated data items. The number of fixed replicas is determined according to the number of areas and sub areas that are covered by the replication system.

3.2 Replication Model

To act in accordance with characteristics of LMDDBSs with large number of updates, an abstract replication model for such systems is specified in this thesis. This model is based on the well-known concept of the Publisher-Subscriber relation. According to this relation, the Publishers concept is used to represent the hosts that generate updates to be propagated to other hosts that act as subscribers to those generated updates. In our replication system, the host can perform the two roles as shown in figure 3.2.1 where both hosts A and B make their updates available to each other.

In this replication model, replicated data are classified into frequently changed data and infrequently changed data. The update of infrequently changed data is allowed only at a single publisher, while update of frequently changed data can be performed on any publisher. The restriction of updating infrequently changed data to a single host comes from the fact that the objects that store these data have relationships with at least one of the other objects that hold either frequently or infrequently changed data. For example, the object that contains the disease details may have a relationship with the object that contains follow-up data of patients in that they are linked based on the values of *Disease-Code* attribute. If we allowed the update on Diseases object to be occur in each publisher, this leads to inconsistent values of Disease-Code (the same disease has multiple codes). Accordingly, this restriction ensures that each disease has one code that is stored on all publishers.

Thus, each host in the replication system can act as a publisher and a subscriber for updates that occurred in the frequently changed data, while only one host acts as a publisher for updates occurred on infrequently changed data and all other hosts act as subscribers for such type of updates.

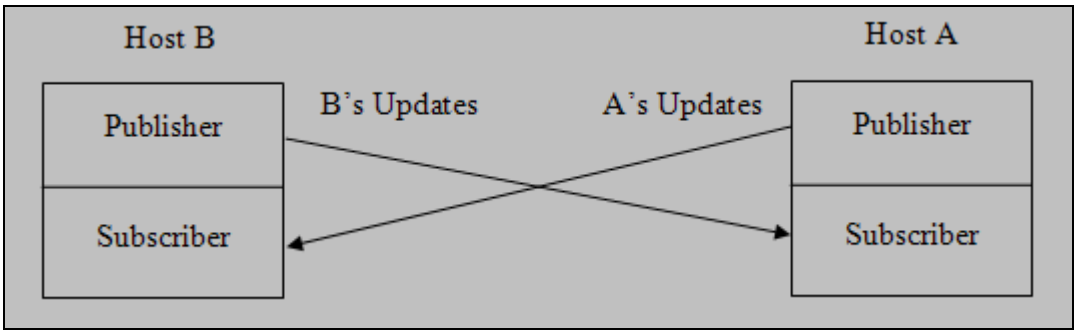

Figure 3.2.1 Publisher-Subscriber Relationship between Two Hosts A and B

The replication model encompasses a hybrid of both Client/Server replication and Peer-to-Peer replication in order to benefit from their advantages. The Client/Server replication is exploited in ensuring that updates occurred on any publisher will be propagated to all subscribers that contain same data objects in other geographic areas. Also, it is exploited in allocating replicas to new publishers that join the replication system, and ordering of updates that occurred on different publishers that act as clients. The Peer-to-Peer replication is exploited in exchanging recent updates between nearby publishers in the same geographic area.

This abstract model defines five phases for the replication process in LMDDBMs (see figure 3.2.2) as follows.

1. Initial Replication

In this research, the initial replication refers to the allocation of replicas of the database to the hosts that have joined the replication system. Each new added host to the replication system receives a copy of the database from the server that exists in the higher level. This copy contains both the database schema and the data that are stored on the database. The mobile host receives the copy from the server of the cell where it is currently located. For each host, the received copy contains only selected objects of the database that are needed for the user to perform his duties, while the received copy by a server represents a full replica of the database (i.e. all objects are replicated on the servers), since each server participates in the initial replication by providing new hosts in its area with an option to select their required data objects. Moreover, the replicas that are stored on the servers are continuously up to date because these servers have stable connectivity among them.

Accordingly, the replica that is stored in each sever is called *Trusted Replica*, as from it the hosts can copy their interested objects. This phase is executed at infrequent intervals, since it is needed only to store a local replica of the database on new hosts or when the host desires to change the set of replicated objects.

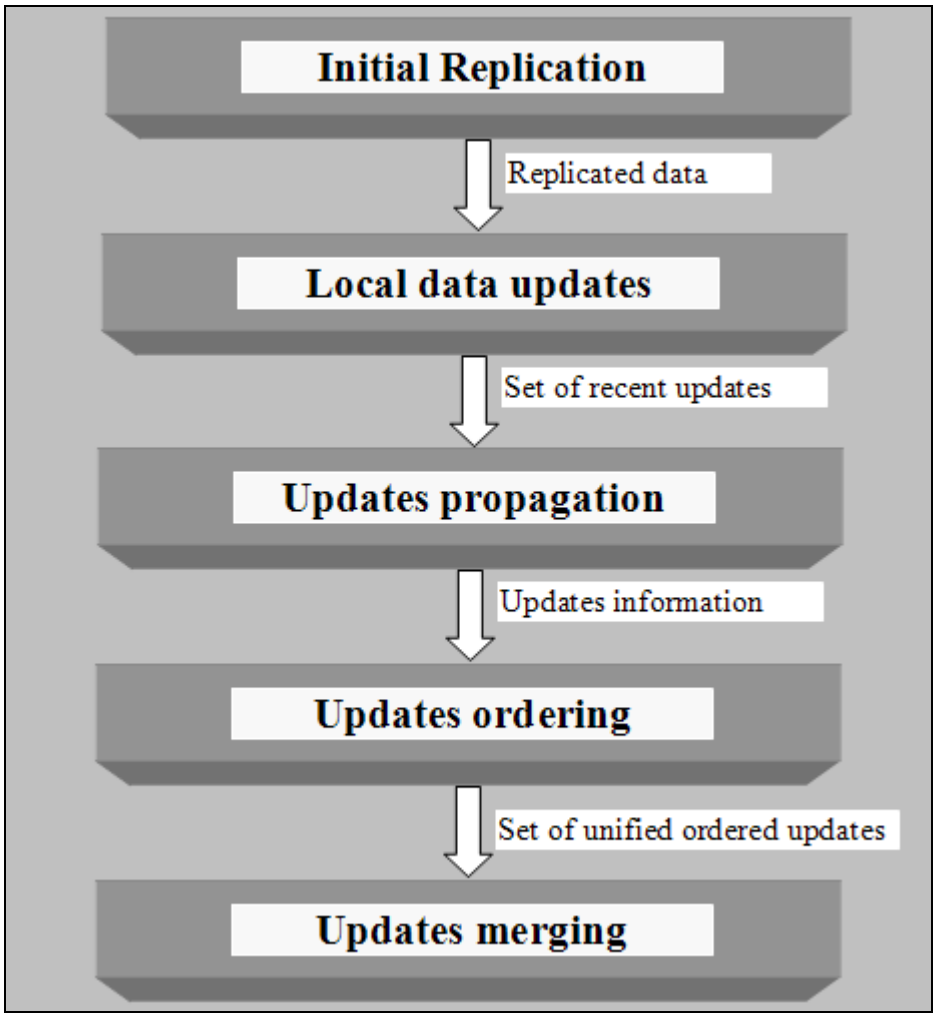

Figure 3.2.2 The Five Phases for the Replication Process in LMDDBMs

2. Local data updates

In this phase, update operations are performed locally on each replica (i.e. publisher) without needing to contact any other replica. This means that all operations submitted by the locally running application on a specific host are committed locally as stable updates

that represent the current changes of the data in the system, which is represented by the database. This phase is repeated frequently in each publisher according to the occurrence of local updates.

3. Updates propagation

The replicas enter this phase when they are connected with each other. They exchange the information of all recent updates among them. Typically, the replica in this phase performs both publisher and subscriber roles regarding the propagation of frequently changed data. It propagates (i.e. publishes) its local updates, while receiving updates occurred on other replicas.

4. Updates ordering

This phase is performed on specific publishers that represent the servers located in the fixed network. These publishers act as points of collection of updates from other publishers (i.e. hosts) that located in their areas, which send their updates to those specific publishers in order to ensure that these updates will be published (disseminated) to all other replicas. The specific publisher orders the collected updates according to specific criteria that ensure a unified ordering for these updates before sending them to the higher level. The phase is repeated upon collecting updates from underlying publishers.

5. Updates merging

In this research, the merging process means inserting the received updates from other publishers in the local database and ordering existing updates accordingly. This phase is performed on each subscriber upon receiving of unified ordered updates from those specific publishers (i.e. publishers perform updates ordering). These updates are merged with the updates that occurred in the received publisher in the same order as received from the specific publishers.

3.3 Replication Architecture

The proposed replication architecture (see Figure 3.3.1) considers a total geographic area called the master area that has a server called Master Server (MS) and a set of fixed hosts. The master area is divided into a set $Z = \{z_1, ..., z_n\}$ of zones. Each zone has a server called Zone Server (ZS) and a set of fixed hosts and it consists of a set $C = \{c_1, ..., c_m\}$ of smaller areas called cells. Each cell represents an area, where the mobile users can perform their duties at a particular period of time before moving to another cell. Each cell has a server called Cell Server (CS). In this architecture, the network is divided into fixed network and mobile network. The fixed network consists of fixed hosts and wired local area network to connect them in the master area. Also, it includes wide area network to connect the master server with zone servers, and to connect zone server with underlying cell servers. The cell server is augmented with a wireless interface and acts as a mobile support station for connecting mobile hosts to the fixed network. On the other hand, the mobile network consists of wireless network and mobile hosts in the cell area.

To provide more flexibility and application areas for this architecture, replicas are divided into three levels:

Master Level: In this level, the replica that is stored on the master server must be synchronized with replicas from the zone level. The master server is responsible for synchronizing all changes that have been performed on both infrequently changed data and frequently changed data with the lower level.

Zone Level: In this level, each replica must be synchronized with replicas from the lower level. The zone server is responsible for synchronizing all intra-level changes with the master server.

Cell Level: Each replica in this level is updated frequently, and then synchronized with the cell server's replica and in turn the cell server synchronizes all intra-level data with the zone server.

In this architecture, initially, the database is stored on the master server. When dividing the master area into multiple zones, a replica of that database is distributed to zone servers. Similarly, new replicas are created when dividing the zone area into multiple cells to represent the cell servers and when registering new mobile hosts and fixed hosts in the replication system. The information of each new replica is stored on the

host's object in the server of the area where the replica is created and then it is replicated to other servers. This information includes the *Host ID*, *Host Type* (FH, MH, CS, ZS, and MS), and *Region* where it is registered. The details of the replicated objects on that host are stored on the object of host's replicated data.

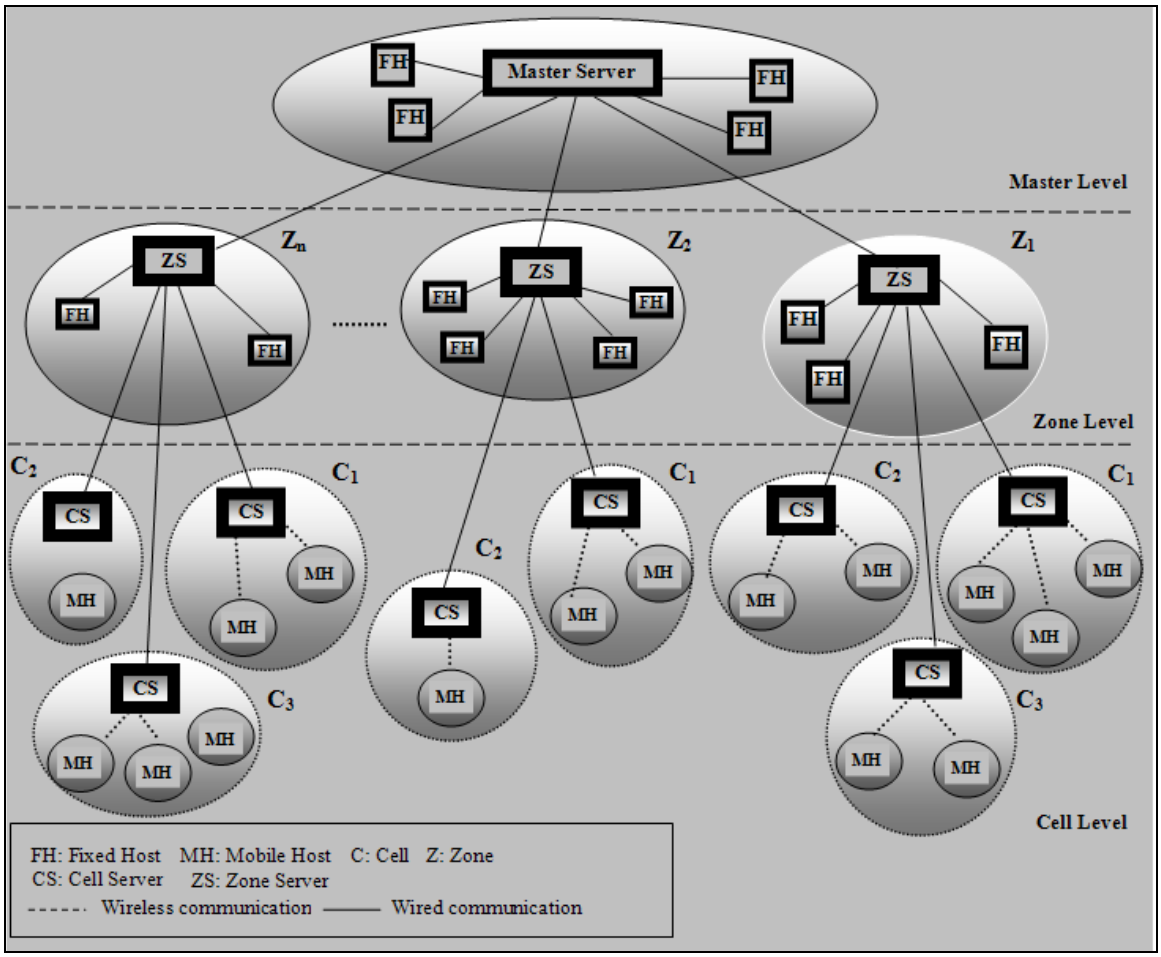

Figure 3.3.1 The Replication Architecture for Large-Scale Mobile Environments

3.4 Replication Method

The replication method is based on a new type of software agent called Instance-Immigration-Removed Agent (IIRA) that is introduced in this research. The research chose this name, according to IIRA working nature, since it creates an instance of itself and this instance migrates to another host and performs its task before removing itself.

The purpose of the instance is to propagate recent updates that occurred or are collected in one level to another level in the replication architecture.

The following definition formally defines IIRA.

Definition 3.4.1 IIRA is a 5-tuple $< I, D, T, S, U >$, where:

$I = \{i_1, \ldots, i_n\}$ is a finite set of primitives/instructions that are required to perform IIRA tasks.

$D = \{d_1, d_2, d_3, d_4\}$ is a finite set of data structures that are involved in propagating recent updates between the components of the replication architecture, collecting recent updates from underlying levels, and initial replication process.

$T = \{t_1, t_2, t_3, t_4\}$ is a finite set of IIRA types. A type t_i maps each IIRA to a certain level (i.e. the type determines the location of IIRA).

$S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ is a finite set of IIRA states. Each state represents the current activity that is carried out by IIRA.

$U: T \rightarrow \{1, 2, 3, \ldots, k\}$ is a function for assigning a unique identifier for IIRA in the system.

According to the abovementioned formal definition, the IIRA consists of code and database and it has type, state, and unique identifier.

The IIRA resides with the application, the replica manager, and the replicated database in its environment (i.e. home host) as shown in Figure 3.4.1. The IIRA dwells with the application on the top layer over both the database system and the Host Manager layers. It interacts with the database for retrieving the set of recent updates that are generated according to the accesses that are performed by the application, and inserting the set of updates that occurred in the other hosts. Also, IIRA interacts with the Host Manager, which represents a software that is running at all times on the host to control the operations and usage of all devices attached to the host, and execution of different applications. Interaction between IIRA and the Host Manager is realized through message passing for requesting the Host Manager to transfer its instance to the other host that the connection is realized with, and for monitoring the connection with the other hosts to determine which host is currently connected with its home host.

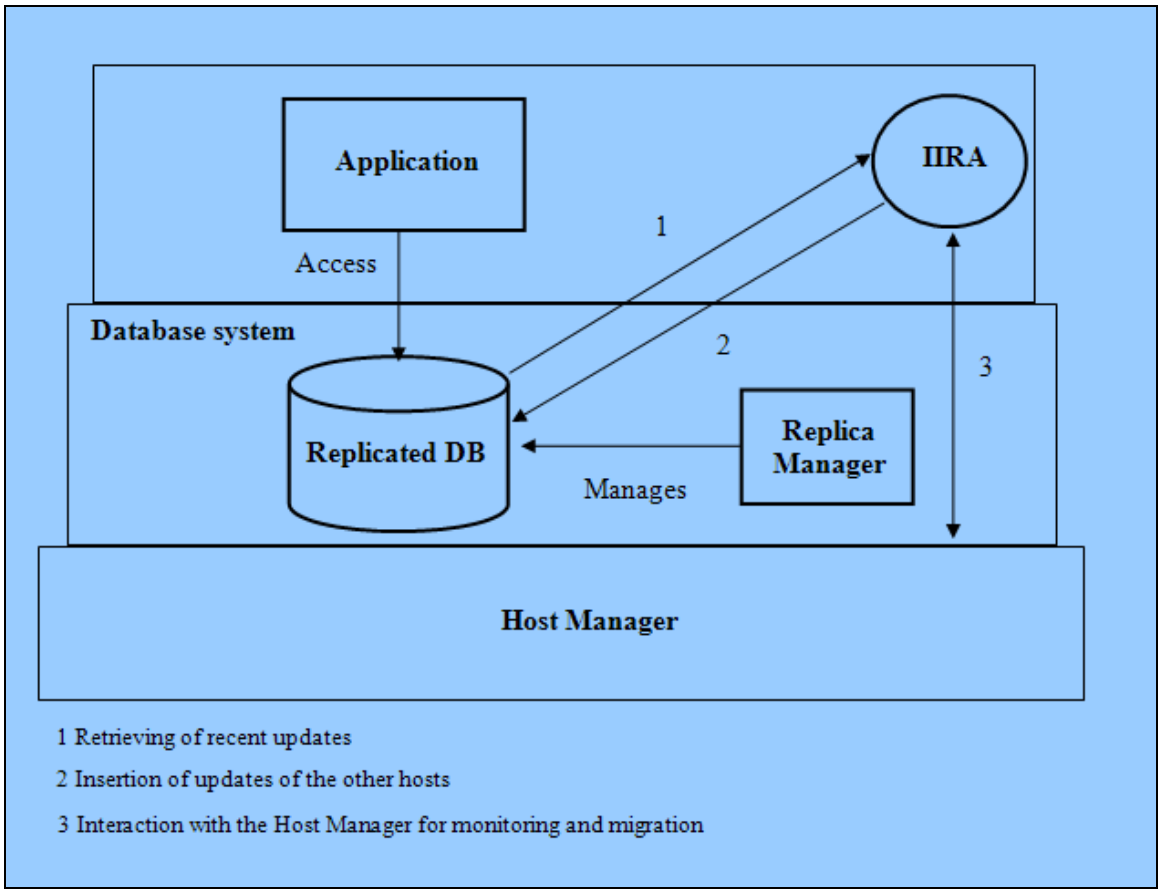

Figure 3.4.1 The Location and Interactions of IIRA in its Environment

3.4.1 The Structure of IIRA

The IIRA's code contains the required instructions for performing IIRA tasks, which include communicating with its host, initial replication process, retrieving recent updates, and propagating updates to the other hosts. The database part of IIRA represents a collection of data structures that are involved in propagating recent updates to the higher level, propagating unified ordered updates to the lower level, collecting recent updates from underlying levels, and performing initial replication process. The data structure stores temporary information that related to updates that either occurred or collected on the host.

These data structures are called View-Like data structures (VLDS) according to their structures. VLDSs contain two sections. The first section is a modified row(s) that is filled by IIRA with the details of the instance(s) such as the ID of the instance, ID of the host, the value of the timestamp when the instance is sent, and the ID of the destination. The second section contains the updates information that is either retrieved from the objects in the replicated database or collected from underlying level. Updates in this section are grouped according to each object. The first section is called the Header, which contains the details of the instance and the second section is called the Body, which contains the information of recent updates. The contains in each section are temporary because they are removed from VLDS once they are received by the other host. The following figure depicts the general structure of VLDS.

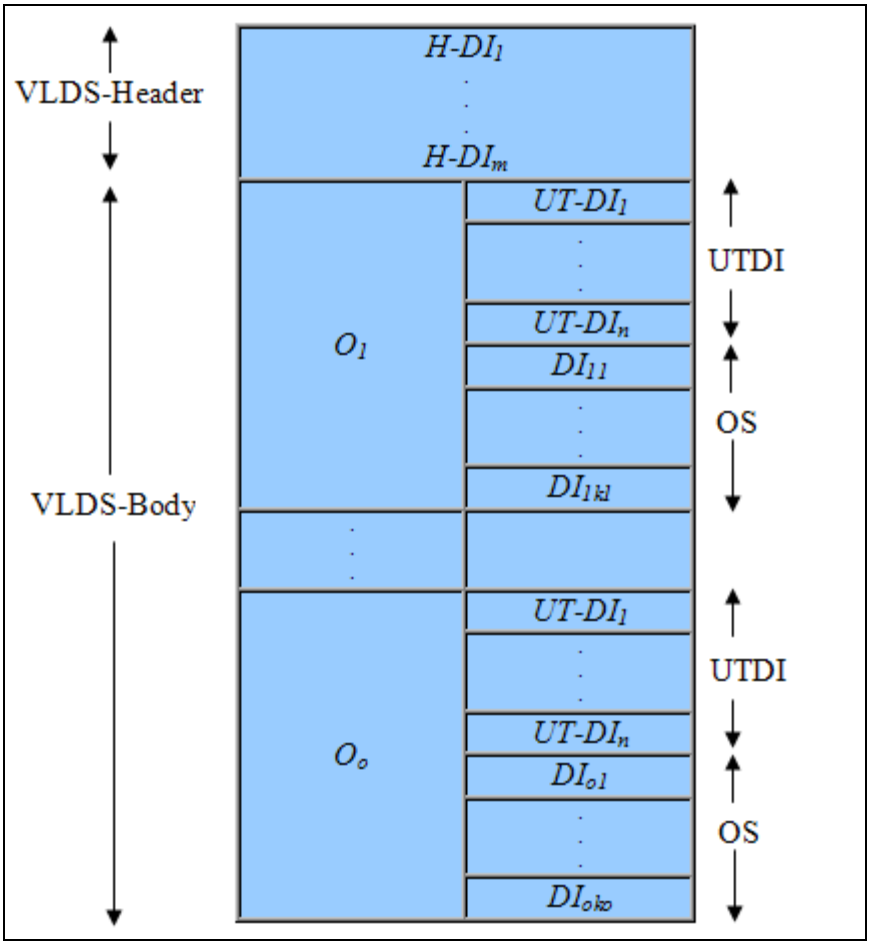

Figure 3.4.1.1 The General Structure of VLDS

The notations that are used in this figure are described in the following table.

Table 3.4.1.1 VLDS Notations

Notation	Meaning
UTDI	Updates Tracking Data Items
OS	Object's State
O	Object
o	Number of objects that affected by updates
k_i	Number of data items replicated in object *i*.

The description of data items in this figure is as follows.

- $H\text{-}DI_i$ (i=1,…, m) is the number of the data item that is used as an attribute of the instance. Examples of such data items include the ID of the sender of the instance, the ID of the Instance, etc.
- *UT-DIi* (i=1,...,n) is the number of a data item that is associated with each update in order to facilitate the ordering process in the higher level. Examples of such data items include the number of update, the value of the timestamp when update is generated, and the value of the timestamp when update is sent to the destination. Also, this data item can represent an attribute to an ordered update that has been ordered by the IIRA in the highest level and is propagated to lower levels. An example of this data item is the value of the *Global ID*, which is assigned to each ordered update. The value of n is different from a level to another level, since the number of these data items increases as updates information moves from the lowest level to the highest level (n for MH < n for CS < n for ZS < n for MS) and it becomes fixed as updates information moves from the highest level to the lowest level.
- DI_{ij} (i=1,…, o and j=1,…, k_i) is the data item number j in the object number i. The total number of objects that are replicated on a host is o and the total number of data items that are replicated on each object is k_i.

VLDSs are divided into propagation data structures (Propagation-VLDSs), collection data structures (collection-VLDSs), and initial replication data structures. This classification according to the type of the operation that is carried out by IIRA, which is propagating recent updates, collecting updates for ordering process, or performing the initial replication process for allocating replicas. Moreover, Propagation-VLDSs are divided into two types based on the category of updates that should be propagated to the other level. Two categories are defined for updates. The first category is the Recent Updates (RU), which contains updates that are generated or collected on a given level and will be propagated to the higher level in order to be available for other replicas. The second category is the Recent Resolved Updates (ReRU), which contains updates that are totally ordered in the highest level and will be propagated to lower levels. Accordingly, the four types of VLDSs are as follows.

a. **RU-Propagation-VLDS.** This type stores the set of recent updates that occurred or are collected in a given host in order to propagate them to the host in the higher level.
b. **Collection-VLDS.** It stores the set of updates that are received from underlying level in order to collect them for the ordering process.
c. **ReRU-Propagation-VLDS.** This type stores the set of updates that are totally ordered by MSR-IIRA (see the following section) to propagate them to underlying level until they reach MHs.
d. **Initial-Replication-VLDS.** It stores the information of the replicated objects on a new host.

The identified types of VLDSs are distinguished in their structure based on the number and values of data items that the type has in both header and body sections. Moreover, each type is customized according to the type of the host (MH, CS, ZS, and MS) based on same criteria (i.e. number and values of data items).

3.4.2 IIRA Types

The research divides IIRA into four types according to the level in which the IIRA carries out its activities. The four types share common structure and behavior, but they inhabit different levels. These types are as follows.

1. **MH-Resident IIRA (MHR-IIRA)**
   Every MH has IIRA, and it is responsible for propagating recent updates that are performed on MH to the other hosts in the mobile network or to the cell server that covers the cell where the MH is located at the time of connection.
2. **Cell Server-Resident IIRA (CSR-IIRA)**
   This type acts as a broker for propagating recent updates between the fixed network and the mobile network. It propagates all updates that are received from MHR-IIRA to the zone server and vice versa. It is responsible for providing a unified causal ordering for all updates that occurred in its cell and maintaining total ordering of messages that are received from the higher level.
3. **Zone Server-Resident IIRA (ZSR-IIRA)**
   This type receives all updates that are performed in the fixed network and the mobile network, which are associated with specific zone level, and provides a unified causal ordering for these updates on this level. Then, it propagates these updates directly to the master server. Also, this type provides underlying cell servers with a set of totally ordered updates that is received from the master server.
4. **Master Server-Resident IIRA (MSR-IIRA)**
   This type receives all updates that are performed in the zone level and provides a total unified ordering for all updates occurred in the replication system. Then, it propagates these ordered updates directly to each zone server, which in turn propagates these updates to underlying levels.

3.4.3 IIRA States

The possible states (i.e. activities) of the IIRA in the replication system are as follows.

(i) **Monitoring:** IIRA monitors the connection with the other hosts though interacting with the Host Manager via message passing. IIRA performs this activity upon the migration of its instance to the other host. It requests the Host manager through a message to inform it in case of any connection is realized with another host.

(ii) **Retrieving:** If the connection is established with another host, IIRA retrieves the set of recent updates that are either occurred or are collected in its host. IIRA recognizes recent updates based on the values of Propagation-Flag data item. Thus, IIRA enters this state when the monitoring state results in a connection that is realized with the other host.

(iii) **Creating the instance:** IIRA creates an instance of itself that contains:

- **(a)** The code that is required for performing both insertion of updates and removing of the instance activities on the other host.
- **(b)** A data structure for shipping updates, which is either RU-Propagation-VLDS or RU-Propagation-VLDS according to the level of the other host and the type of the propagation.

(iv) **Migrating:** IIRA requests the Host Manger to transfer its created instance to the other host that the connection is realized with it. If the Host Manger transferred the instance successfully, it informs the IIRA through an acknowledgement message. Otherwise, IIRA requests the Host manager once again until it receives the acknowledgement message.

(v) **Executing on the other host:** Upon arriving at the other host and obtaining the permission for execution from its Host Manager, the migrated instance inserts its stored recent updates in the database of the IIRA in the other host (i.e. For RU propagation) or in its replicated database (i.e. for ReRU propagation).

(vi) **Removing:** The migrated instance removes itself after completion of the insertion process.

Figure 3.4.3.1 shows the above activities that are carried out by IIRA and its instance in both the home host and the destination.

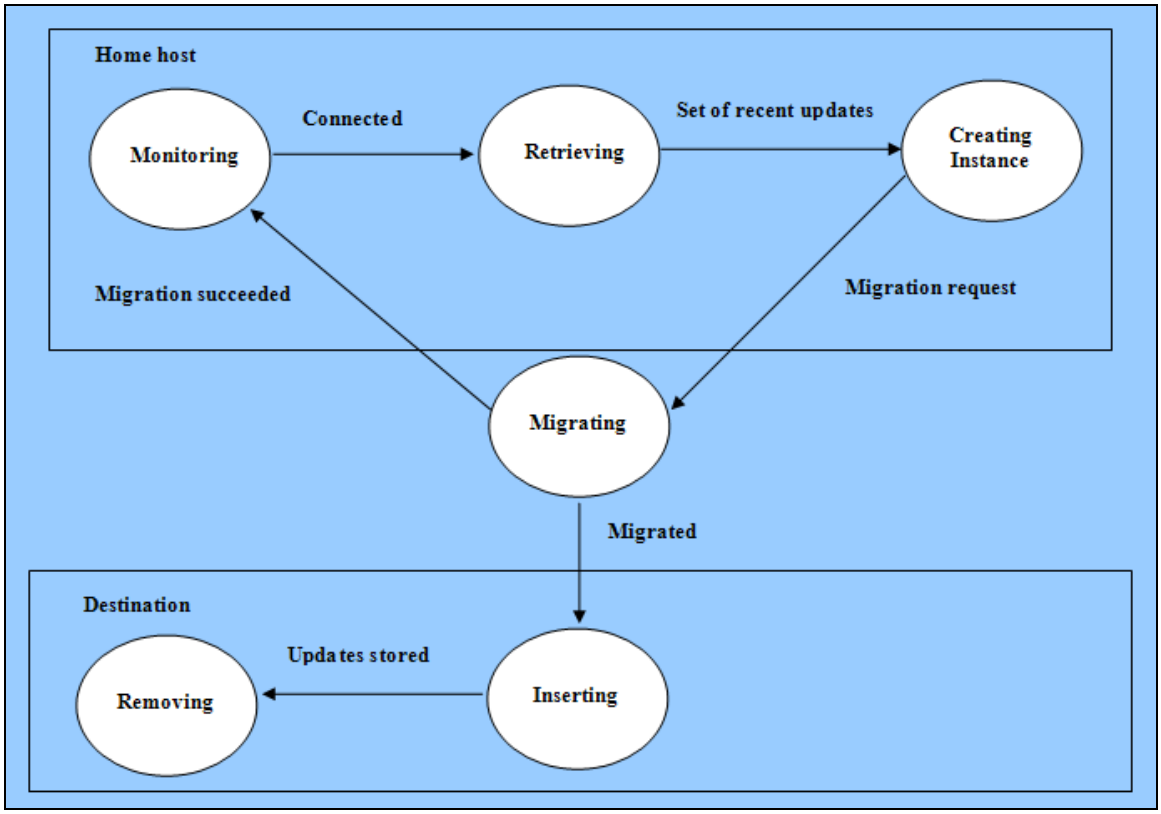

Figure 3.4.3.1 State Diagram for IIRA and its Instance

3.5 Initial Replication

As aforementioned, the initial replication refers to the process of allocating replicas to new hosts that join either the fixed network or mobile network in the replication architecture. This process is performed by the IIRA type that exists in the higher level. MSR-IIRA allocates replicas from the master database to the new zone servers. Then, each ZSR-IIRA allocates a full replica to each one of its underlying cell servers. And in turn each CSR-IIRA distributes partial replicas to the mobile hosts that connected with it.

For this process to take place, the new host should be registered first through issuing a *join* request (which is a message that is sent from the new host) to the host in the higher level. Once the request is verified and accepted by the host in the higher level, a message is sent to the IIRA in this host (i.e. the server in the higher level). The IIRA

creates an entry in the *Hosts-Obj* object for the new host. Then, an instance of the IIRA is created. This instance is called *IIRA-Initial-Replication-Instance* which contains the code that required for creating the database schema and *Objects-Information-VLDS*, which contains the objects details that are retrieved from *Objects-Details-Obj* object (see appendix A), which stores the details of the objects and their data items. Then, this instance migrates to the new host.

The steps that are carried out by the migrated instance for performing the initial replication vary according to the type of the new host. If the type is either mobile host or fixed host, there will be a need to determine the data of interest to the new host (i.e. partial replication). Therefore, the migrated instance will perform the following steps when it is executed on the new host:

Step 1 *(Determining the objects and data items of interest to the new host)*

1.1 Informing the user through an interface to select the objects and data items that will be replicated to his host.

Step 2 *Creating the replicated database on the new host according to those selected objects and data items by the user.*

S***tep 3*** *The instance creates an instance of itself to be the IIRA of the new host, which stores only the code and VLDSs that are required for updates collection and propagation processes.*

Step 4 *Storing the information of the replicated objects and data items on the Hosts-Replicated-Objects-Obj object.*

Step 5 *(Informing the IIRA in the higher level about the replicated objects for the new host)*

5.1 *The migrated instance retrieves the data on Hosts-Replicated-Objects-Obj object into its Replicated-Objects-Information-VLDS.*

5.2 *Creating an instance of itself.*

5.3 *The created instance returns back to its parent host in the higher level.*

5.4 *Executing on its parent host through inserting the contains of the Replicated-Objects-Information-VLDS in the Hosts-Replicated-Objects-Obj object in the replicated database.*

Step 6 *IIRA in the higher level creates an ordinary instance of itself to propagate all data that stored in the objects and data items that have been selected by the new MH to it.*

Step 7 *The IIRA in the higher level sends the same migrated instance from the lower level to the host in its higher level to propagate the information about the new host and its replicated data.*

Step 8 *Repeating step 7 until these information reach the highest level in order to be propagated to all underlying hosts through ordinary instances that hold RU-Propagation-VLDS.*

In the case of the user desires to change its replicated objects, the following steps will be carried out.

Step 1 *The user invokes a service attached to its local IIRA called Change-Replicated-Data.*

Step 2 *Once the service is invoked, the local IIRA will continue on monitoring the connection with the host in the higher level.*

Step 3 *If the connection is realized, IIRA will send a message called Change-Replicated-Data-Msg to the IIRA in the higher level.*

Step 4 *Upon receiving the message, the IIRA in the higher level will create IIRA-Initial-Replication-Instance, then this instance migrates to the calling MH.*

Step 5 *When executing on the underlying host, the instance performs the abovementioned steps from 1 to 8 except here it will add or remove only a certain number of objects or data items.*

If the type of the host is either Cell server or Zone server, this means that all objects will be replicated to the new servers because they replicate the database in a full replication manner. Accordingly, migrated instance from the higher level will perform the following steps.

Step 1 *Creating the replicated database on the new server.*

S*tep 2* *Creating an instance of itself to be the IIRA of the new server, which stores the code and VLDSs that are required for collecting updates, propagating updates, and initial replication processes.*

Step 3 *Storing the information of the replicated objects and data items in the Hosts-Replicated-Objects-Obj object.*

Step 4 *Informing its parent IIRA in the higher level that the initial replication is performed successfully.*

Step 5 *Removing itself.*

Step 6 *The parent IIRA stores the information of all objects for the new server in Hosts-Replicated-Objects-Obj object in the replicated database.*

Step 7 *Creating an ordinary instance of itself to propagate all data that stored in all objects to the new server.*

Step 8 *Sending an instance to the host in the higher level to inform it about the new server through an ordinary instance that contains RU-Propagation-VLDS, which ship only one record that is retrieved from the Hosts-obj (there is no need for shipping the details of replicated objects for the new server because each server is considered that it holds all objects).*

Step 9 *Repeating steps 7 and 8 until this information reach the highest level in order to be propagated to all underlying hosts through ordinary instances that hold RU-Propagation-VLDS.*

3.5.1 Initial-Replication-VLDS

Two sub types of this type are involved in the initial replication process. These subtypes are as follows.

(i) Objects-Information-VLDS. It is stored in the instance that will migrate to a host in underlying level for the purpose of performing the initial replication process. This type contains only the body section as shown in the following figure.

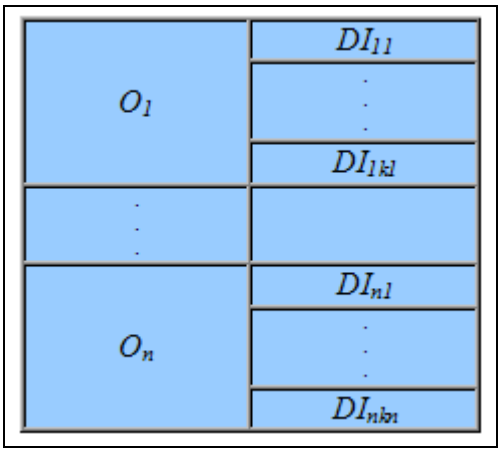

Figure 3.5.1.1 The Structure of Objects-Information-VLDS

In this figure, DI_{ij} (i=1,…, n and j=1,…, k_i) is the data item number j in the object number i. The total number of objects replicated in a host is n and the total number of data items replicated in each object is k_i.

When the instance executes on the destination, it displays the list of objects to the user by retrieving the data that are stored in this VLDS. According to the user's selection, it executes the required statements for creating objects.

(ii) Replicated-Objects-Information-VLDS. It is used to ship the information of the replicated objects on a new host, which is retrieved from *Hosts-Replicated-Objects-Obj* object. This type is shown in the following figure.

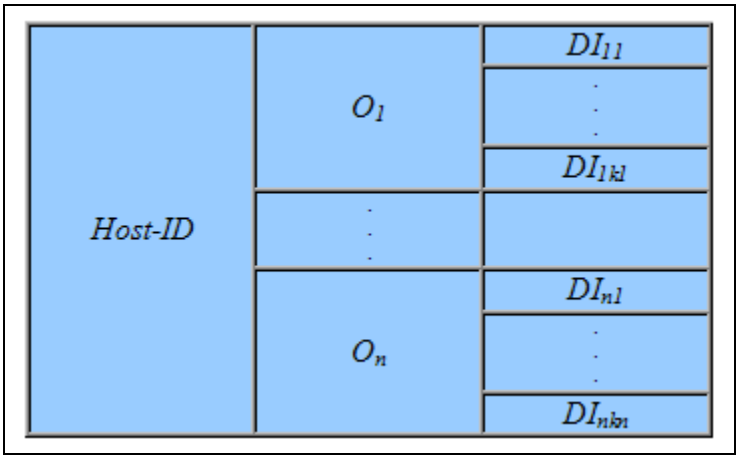

Figure 3.5.1.2 The Structure of Replicated-Objects-Information-VLDS

An important concept in the initial replication is *Object-Related-List*, which refers to all related objects to those that have been selected by the new host in the initial replication process.

Definition 3.5.1.1 *Object-Related-List* is a set of data objects that contains only infrequently changed data and have relationships with other data objects through an attribute or a set of attributes.

According to the replication model, each object belongs to *Object-Related-List* can be updated only on the master server (i.e. the only one publisher for updates occurred in infrequently changed data).

Thus, in the initial replication, when the new host selects the objects that wants to replicate, the IIRA looks at *Object-Related-List* to determine which objects in this list related to the selected objects. Then, it replicates these objects into the new host by adding them to the objects that are selected by the host.

3.6 Summary

In this chapter, the specifications of the proposed replication strategy are outlined, and its components and their purposes are stated. The strategy encompassed four components in order to act in accord with the characteristics of LMDDBSs. A description of the replication architecture and the replication method is provided. The description of the other two components (i.e. updates propagation protocol and updates ordering mechanism) is provided in chapter 4 and chapter 6, respectively. Also, this chapter presented the formal system model and the abstract replication model for LMDDBSs, and the details of replicas allocation.