# CHAPTER FIVE

## IMPLEMENTATION OF UPDATES PROPAGATION PROTOCOL USING IIRA-BASED PROPAGATION SYSTEM

### 5.0 Overview

This chapter provides a description of IIRA-based propagation system. The purpose of this system is to implement the proposed propagation protocol through automating the propagation of updates between the components of the replication architecture in a manner that ensures bounding unavailability of recent updates and inconsistency of replicated data, and propagating only recent updates among the large number of replicas. The chapter presents the components of this propagation system and the interaction mechanisms between them. Also, it presents the operations that are carried out by IIRA with respect to the different types of updates propagation.

To model and analyze the stochastic behavior of the proposed propagation system with regard to reaching a mobile database state in which availability and consistency are satisfied, the research developed Stochastic Petri Net (SPN) model. The Continuous Time Markov Chain (CTMC) is derived from the developed SPN and the Markov chain theory is used to obtain the steady state probabilities of occurrences of the different states of the replication system.

### 5.1 IIRA-Based Propagation System

The IIRA-Based propagation system (see Figure 5.1.1) is composed of the four types of IIRA. Each type interacts with the other types through a synchronization process in which two types exchange their recent updates via their created instances of IIRA. In this system, each type can exchange updates directly with the same type or a different type in the server of the level where it inhabits or in a server from underlying level. For example, MHR-IIRA can exchange updates directly with CSR-IIRA or another MHR-IIRA, while MHR-IIRA cannot directly interact with either ZSR-IIRA or MSR-IRRA.

When the connection takes place between any two IIRA types, an instance of IIRA type that inhabits the lower level will propagate a set of recent updates called Updates-Result (UR), which is produced by an application, in addition to updates that may have been collected from the underlying level to the database of IIRA type that inhabits the higher level. Then, an instance of the type that inhabits the higher level propagates a set of updates called Recent-Resolved-Updates (ReRU) that is received from the higher level to the replicated database in the lower level. Accordingly, IIRA has different responsibilities for updates propagation with respect to implementing the three basic mechanisms of updates propagation (Bottom-Up Propagation, Top-Down Propagation, and Peer-to-Peer Propagation). These responsibilities are identified in section 5.1.2.

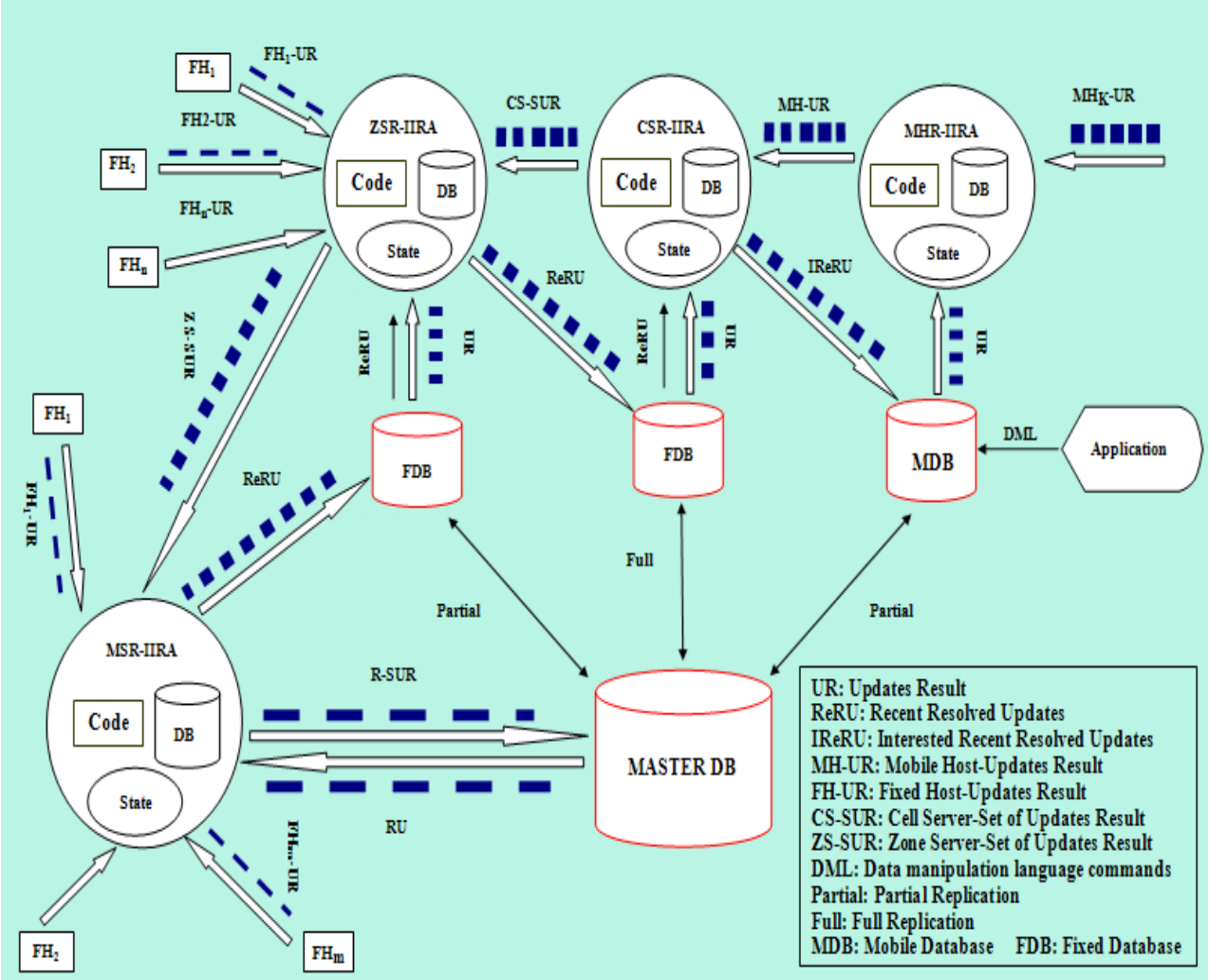


**Figure 5.1.1 IIRA-Based Propagation System**

### *5.1.1 Waiting Period for Connection*

The exchanging of updates between two types occurs only during the connection period between their home hosts. The time period that a mobile host waits for the connection with a cell server in the fixed network is not deterministic and it depends on its own conditions, such as connection availability, battery, etc. On the other hand, the research assumes that the time period that a server in the fixed network must wait to connect with the server in the higher level is deterministic and its value depends on the availability and consistency requirements of the replication system. To decide this value, the research views the connection timing system for connecting servers in the fixed network should mimics the shifting behavior, where the shift time is exploited in collecting recent updates from underlying level (e.g. the cell server should connect to the zone server every one hour to propagate the collected updates from underlying mobile hosts during the last past hour).

### *5.1.2 IIRA Responsibilities for Updates Propagation*

The responsibilities of IIRA can be identified according to the mechanism of the propagation as follows.

**Bottom-Up Propagation:** In this type, each MHR-IIRA propagates the set of recent updates (MH-UR) that occurred in its host to the database of the type that inhabits a cell server. Also, each server's IIRA collects the received recent updates from underlying level in addition to its server's updates-result in a set called Set of Updates Result (SUR), and resolves updates conflict through ordering updates in this set, and then propagates this set to the database of IIRA in the higher level.

As previously mentioned, the time period that the server in the current level should wait for receiving recent updates (i.e. updates collection) from underlying level is deterministic. The value of this period in a server in the higher level is greater than its value in a server in the lower level (e.g. waiting period for the zone server > waiting period for the cell server). This is because the number of hosts, where updates occurred increases as we move to the higher levels. After elapsing of this deterministic period, the IIRA carries out the required operations for this type of propagation.

The typical operations that are involved in this propagation, which are performed by IIRA are:

- Resolving updates conflict through ordering the collected updates, which are stored in Collection-VLDS on the database of the IIRA type that inhabits the host in the current level.
- Assigning the value of the update timestamp data item (in case of the update is generated on the same host).
- Creating the instance and filling the RU-Propagation-VLDS with the ordered updates that are retrieved from Collection-VLDS.
- Migration of the instance to the host that exists in the higher level.
- Assigning the value of the send timestamp data item to each update before the migration start and after migration request is accepted, and assigning the value of receive timestamp when the instance arrives at the destination.
- Execution of the instance in the destination host through insertion of recent updates in its IIRA type's database.

Thus, this propagation takes place in the servers that exist in the fixed network when IIRA orders the collected updates in Collection-VLDS. Then, the ordered updates are shipped via RU-Propagation-VLDS to the host in the higher level. This means that updates are pushed from a level to the higher level in a unicasting manner where each host pushes RU to a single host in the higher level. Updates occurred on mobile hosts are pushed to cell servers when the connection occurs.

**Top-Down Propagation:** In this type, each server's IIRA propagates the set of recent resolved updates (ReRU) that is received from the higher level to the lower level. For example, each ZSR-IIRA propagates the ReRU that is received from the master server to underlying cell servers and in turn each cell server propagates a subset of this set called Interested-Recent Resolved-Updates (IReRU) to underlying mobile hosts. IReRU represents the set of updates that are performed on the data items that are replicated on mobile hosts.

The typical operations that are performed by IIRA for carrying out this propagation are:

- Creating the instance and storing the set of recent resolved updates on it.
- Migration of the instance to the host that exists in the lower level.
- Execution of the instance on the destination host.

Thus, for a given set of updates, this propagation takes place when these updates are totally ordered (i.e. resolved) by MSR-IIRA in MS-Collection-VLDS (see section 5.1.3.2). The ordered updates (i.e. ReRU) are then inserted into ReRU-Propagation-VLDS in order to be propagated through the instance to the underlying level. The MS pushes ReRU to all available zone servers through sending same instance to each zone server. Similarly, the zone server does same pushing to its underlying available cell servers. However, the cell servers push ReRU only to connected MHs, while disconnected MHs pull ReRU when they connect to the cell servers. According to this pushing mechanism, this mechanism of propagation can be called Multi-cast Propagation. If some of underlying hosts do not obtained all ReRUs that are propagated previously from the higher level due to some failures, these updates are considered as missed updates. The missed updates are obtained when these hosts connect to the higher level and then pulling such updates from it. This means that the contains of the instance, which will be sent to this host is different from other hosts according to its missed updates.

To determine which ReRUs are missed from a host in the underlying level, the IIRA in the higher level relies on the value of specific data item called *MS-Instance-ReRU-ID* in the *Hosts-Obj* (see appendix One). This data item stores the number of the last instance that has transferred the last ReRU to underlying level. Thus, when the host in the underlying level becomes available, the IIRA in the higher level checks the value of *MS-Instance-ReRU-ID* for that host and fills ReRU-Propagation-VLDS with ReRUs that are missed from it. This filling is performed by retrieving ReRUs from data objects according to their associated *MS-Instance-ReRU-ID* in the *ReRU-Received-Instances-Obj* object (see appendix One).

**Peer-to-Peer Propagation:** In this type, two IIRA of the same type may exchange their updates. The typical operations that are performed by IIRA in this type are same as in Bottom-Up propagation with a distinction that both hosts, which are involved in the synchronization process inhabit the same level and have same type.

### *5.1.3 Implementing Bottom-Up Propagation*

In this section, the required data structures and algorithm for implementing Bottom-Up Propagation are provided as follows.

#### *5.1.3.1 RU-Propagation-VLDS*

The purpose of this type is to store the set of ordered updates that to be propagated from a host in the current level to another host in the higher level. The IIRA fills this type when the collection period for updates is elapsed and the connection takes place with the higher level, while its stored updates are removed when an acknowledgment message is received from the higher level indicating that the updates shipped by this type are received successfully.

The number of data items in the updates tracking part of this type varies according to the type of the host. For example, RU-Propagation-VLDS for the cell server (i.e. CS-RU-Propagation-VLDS) has less data items than RU-Propagation-VLDS for the zone server (i.e. ZS-RU-Propagation-VLDS). Thus, this type has different names and data items based on its host type as follows.

**(i) MH-RU-Propagation-VLDS**

This subtype is stored in MHR-IIRA database. It is used to hold recent updates that occurred in a mobile host to a cell server. The structure of this subtype is as shown in the Figure 5.1.3.1.1.

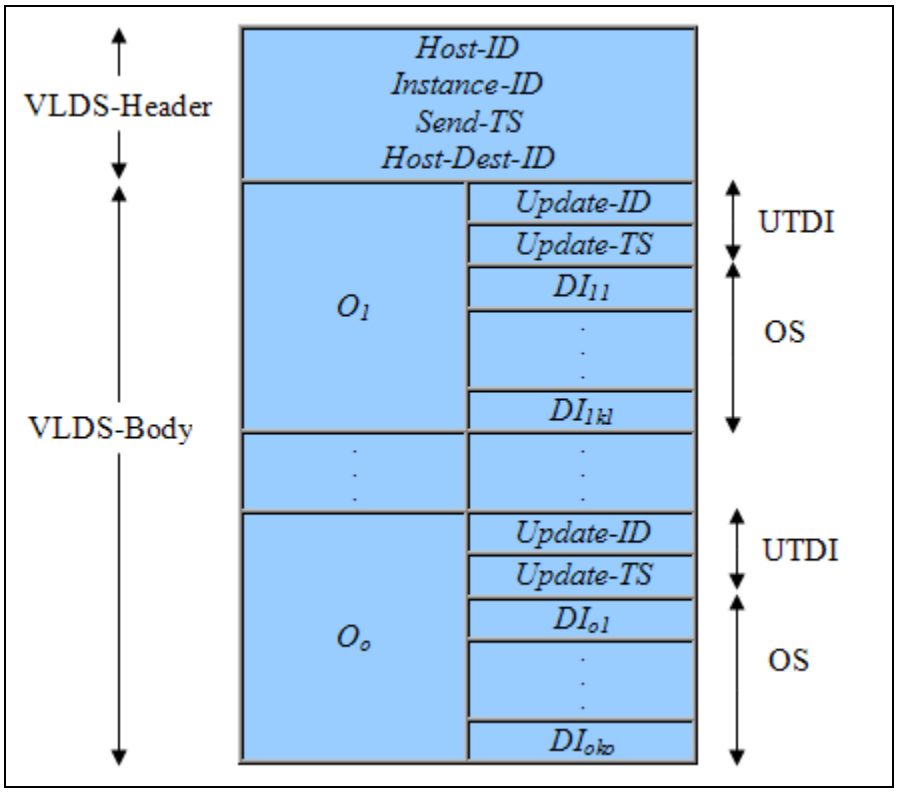


**Figure 5.1.3.1.1 The Description of MH-RU-Propagation-VLDS**

The notations that are used in Figure 5.1.3.1.1 are described in Table 5.1.3.1.1.

**Table 5.1.3.1.1 Notations in Figure 5.1.3.1.1 and Their Meanings**

| Notation | Description |
|---|---|
| *o* | The number of objects where updates occurred |
| *Host-ID* | The number assigned to the source of updates (i.e. Instance's owner) in the Hosts object |
| *Instance-ID* | An incremental number assigned to the instance by IIRA |
| *Send-TS* | The value of the timestamp when the instance is migrated to the host in the higher level |
| *Host-Dest-ID* | The number of the host in the higher level, which the instance will migrate to it |
| *Update-ID* | An incremental number assigned to each update on each object |
| *Update-TS* | The value of the timestamp when the update occurred |

Some constraints are defined on this type of VLDSs, which are:

1. The value of *Host-ID* should be the number of the sending host in the *Hosts-Obj* object.
2. The value of the *Host-Dest-ID* should be either the number of a cell server or the number of another mobile host (in case of Peer-to-Peer propagation of RU).
3. The *Instance-ID* has distinct values. These values are assigned serially based on the following naming schema:

| *Host-ID* | - | *Serial No* |
|---|---|---|

For example, for the instance that migrated from the mobile host that is assigned number 15 in cell number 5, the *Instance-ID* is MH15-C5-942.

4. *Update-ID* is assigned for each new update occurs in an object based on the previous value of the last update occurred in this object.
5. *Send-TS* is the value of *RC* (see section 6.1.2.1) when the instance migrated to the other host.

Two Important notes here are:

- The purpose of the *Update-ID* is as follows. If an update is propagated and then it is modified, this requires that propagating it again. Accordingly, the new update will replace the old one and this depends on the value of this data item.
- The details of the outgoing instances are stored in the *Outgoing-Instances-Obj* object (see appendix One), which represents an archive object that is used to decide the value of the next *Instance-ID* based on the last one.

**(ii) CS-RU-Propagation-VLDS**

This subtype is stored in CSR-IIRA database. It is used to hold an ordered set of both recent updates that occurred in the cell server and collected updates in it from underlying hosts to the zone server. It has the same header as in MH-RU-Propagation-VLDS, but the updates tracking part is different. Figure 5.1.3.1.2 describes the structure of the body in this subtype.

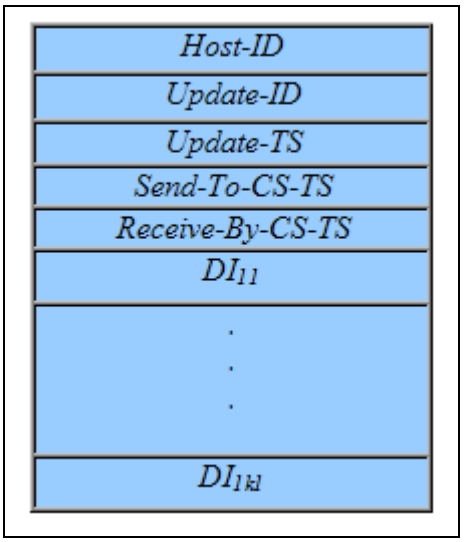


**Figure 5.1.3.1.2 The Structure of the Body Part in CS-RU-Propagation-VLDS**

In Figure 5.1.3.1.2, most notations have same description as in Table 5.1.3.1.1. Additional notations are *Send-To-Cs-TS* and *Receive-By-CS-TS*. The notation *Send-To-CS-TS* is an alias for *Send_TS*, which represents the value of the timestamp when the instance migrated to the cell server. *Receive-By-CS-TS* represents the value of the timestamp when the instance is received by the cell server. The *Host-ID* in the header refers to the sending host (i.e. the cell server), while the *Host-ID* in the body refers to the number of the host that represents the source of the update.

In the header of this type, the value of the *Host-Dest-ID* should be either the number of zone server or a number of another cell server.

**(iii) ZS-RU-Propagation-VLDS**

This subtype is stored in ZSR-IIRA database. It is used to hold an ordered set of both recent updates occurred in the zone server and collected updates in it to the master server. Also, it has different updates tracking part as compared to MH-RU-Propagation-VLDS and CS-RU-Propagation-VLDS. The updates tracking part contains additional two data items than its counterpart in CS-RU-Propagation-VLDS. These data items are *Send-To-ZS-TS* and *Receive-By-ZS-TS*. *Send-To-ZS-TS* is the value of the timestamp when the instance is sent to the zone server. *Receive-By-ZS-TS* is the value of the timestamp when the instance is received by the zone server.

In the header of this type, the value of the *Host-Dest-ID* should be either the number of the master server or the number of another zone server.

#### *5.1.3.2 Collection-VLDS*

This type acts as a pool that encompasses all incoming updates information that is shipped by RU-Propagation-VLDS from underlying level. It stores all received updates during the specified collection period. At the end of the collection period, the collected updates are ordered together with the updates occurred in the same host in the same collection period. Then, the set of ordered updates are propagated through RU-Propagation-VLDS to the higher level.

This type is filled by both incoming instances and the resident IIRA (i.e. It fills this type with the updates occurred in the same host). The header and the body parts of the incoming RU-Propagation-VLDSs are inserted in their counterparts in this type. IIRA empties this type when the collection period is elapsed and its contains are stored in the RU-Propagation-VLDS to be propagated to the higher level.

The IIRA retrieves the local recent updates into this type at the end of the collection period in order to be ordered along with the collected updates from underlying level. For inserting those local updates in this type, IIRA inserts their information by associating them with a virtual instance. In the header of this instance, both the *ID* of the source of the instance (*Host-ID*) and the *ID* of the destination (*Host-Dest-ID*) will be the *ID* of the current host. Also, the values of both *Send-TS* and *Receive-TS* will be same, which take the value of the time instant when the collecting period is elapsed.

The number of data items in the updates tracking part in this type varies according to the type of the host. Thus, this type will has different names and data items based on its host type as follows.

**(i) CS-Collection-VLDS**

It is stored in CSR-IIRA database and is used to store collected updates from underlying mobile hosts and fixed hosts. The structure of this subtype is depicted in the following figure.

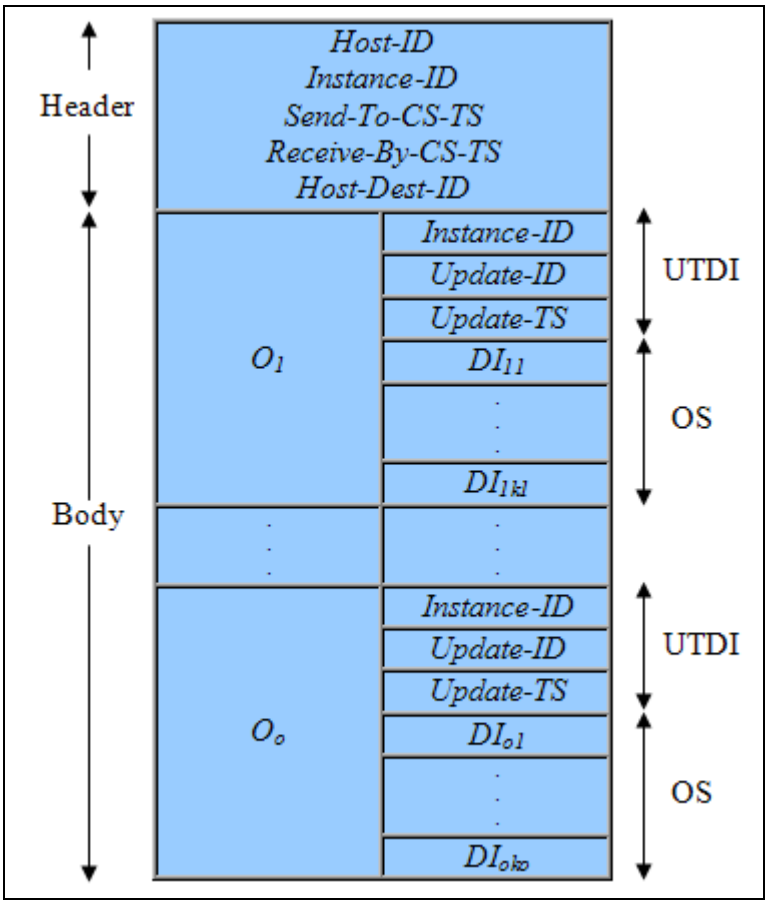


**Figure 5.1.3.2.1 The Description of CS-Collection-VLDS**

In Figure 5.1.3.2.1, each *Instance-ID* in the header has many associated tuples in the body because each instance has many updates. Therefore, the *Instance-ID* is added to the body. *Send-To-CS-TS* is an alias for the *Send-TS* of the instance that is received from the lower level. *Receive-By-CS-TS* is the timestamp when the instance is received by CS. The default value of the *Host_Dest_ID* is the *ID* of the cell server where updates are collected.

**(ii) ZS-Collection-VLDS**

It is stored in ZSR-IIRA database. It stores collected updates from underlying cell servers and fixed hosts. It has the same header as in CS-Collection-VLDS with assigned aliases for *Send-TS* and *Receive-TS*, which are *Send-To-ZS-TS* and *Receive-By-ZS-TS*, respectively. The updates tracking part in the body has additional number of data items. The structure of the body in this subtype is depicted in the following figure.

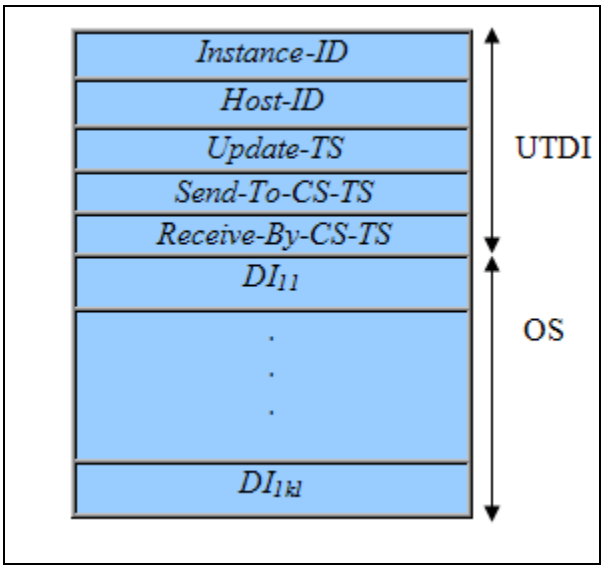


**Figure 5.1.3.2.2 The Structure of the Body Part in ZS-Collection-VLDS**

In Figure 5.1.3.2.2, The *Host-ID* is the *ID* of the host where update is generated, while the *Host-ID* in the header (see the description of CS-Collection-VLDS) is the *ID* of the host that has sent the instance.

**(iii) MS-Collection-VLDS**

This subtype is stored in MSR-IIRA database. It stores all collected updates that are received from underlying zone servers and fixed hosts. Also, it has the same header as in the previous subtypes with assigned aliases for *Send-TS* and *Receive-TS*, which are *Send-To-MS-TS* and *Receive-By-MS-TS*, respectively. The updates tracking part contains additional two data items than its counterpart in ZS-Collection-VLDS. These data items are *Send-To-ZS-TS* and *Receive-By-ZS-TS*.

**(iv) MH-Collection-VLDS**

This subtype can be stored in MHR-IIRA database in the case of MH acts as a broker for many other MHs to propagate their updates to the higher level. This happens when the other MHs cannot synchronize directly with the higher level (e.g. they are in locations far from the location of the server in the higher level). This type has the same structure as CS-Collect-VLDS.

#### *5.1.3.3 Propagating RU Algorithm*

The steps that are carried out in both hosts in the higher and lower levels for propagating RU are as follows.

***Step 0*** *(Verification of elapsing of Updates collection period)*
***If*** *host-type= 'CS Server' or 'Zone Server'* ***then***
***If*** *collection-period elapsed (the time of sending last instance to the higher level + collection-period) ≥ collection-period)* ***then***
*Next Step*
***Else***
*Collect updates*
***Else***
*Next step*

***Step 1*** *(Checking the connection status with the host in the higher level)*
***If*** *the connection is established* ***then***
*1.1 The IIRA retrieves the ordered set of recent updates (RU) from Collection-VLDS into RU-Propagation-VLDS*
*1.2 The IIRA removes the contains of Collection-VLDS*
***Else***
*1.3 Wait for connection*

***Step 2*** *(Creating and populating the instance with the required coda and data)*
*2.1 Creating an instance that contains only RU-Propagation-VLDS and the code that is required for insertion RU in the higher level and removing itself.*
*2.2 Assigning the value of the Send-TS to the instance.*
*2.3 Storing the details of the instance in Outgoing- Instances-Obj object.*

***Step 3*** *Migrating of the instance to the host in the higher level.*

***Step 5*** *(Execution of the instance on the higher level)*
*5.1 Inserting the instance and updates details that are shipped in RU-Propagation-VLDS in Collection-VLDS of the IIRA in the higher level.*
*5.2 Inserting the details of the instance in Incoming-Instances-Obj.*

***Step 6*** *(Finalizing the propagation task)*
*6.1 The instance sends an acknowledgement message to its parent IIRA in the sender indicates that the transferred RU through VLDS is applied successfully.*
*6.2 The instance removes itself.*

***Step 7*** *The IIRA in the lower level removes the contains of the RU-Propagation-VLDS once the ACK message is received.*

#### *5.1.3.4 Updates Ordering*

The ordering process depends on the values of timestamps on both body and header sections on Collection-VLDS as follows.

- In the cell level, if two or more updates have same values for Update-TS data item (i.e. conflicts occurred) in the body section, then the conflicts are resolved based on the value of Send-To-CS-TS in the header section. And if they have same values for Send-To-CS-TS, then the value of Receive-By-CS-TS is used to resolve conflicts.
- In the zone level as well as the master level, when conflicted updates have same values for timestamps in the body section, the conflicts are resolved in the same manner as in the cell level based on the values of Send- TS and Receive- TS in the header section.

After MSR-IIRA performs the total ordering on MS-Collection-VLDS, it assigns Global ID for each update based on the order of that update. The value of this ID represents the order of the update on the level of the whole system. Once the Global IDs are assigned, the MSR-IIRA will store the contains of MS-Collection-VLDS in the replicated database in the master server. Then instances are created to transfer the ordered updates (ReRU) that are stored on this type to the zone servers.

The MHR-IIRA will reorder collected updates from other MHs (in case of they cannot access the higher level) according to the values of the Update-TS, Send-TS, and Receive-TS. Then the result of ordering is sent to the CS using MH-RU-Propagation-VLDS as usual.

### *5.1.4 Implementing Top-Down Propagation*

In this section, the required data structures and algorithm for implementing Top-Down Propagation are provided as follows.

#### *5.1.4.1 ReRU-Propagation-VLDS*

The purpose of this type is to ship ReRU from a level to underlying level until it reaches the lowest level (i.e. the MHs). Accordingly, each server in the fixed network has this type as a part of IIRA's data section. MHs store this type only when they need to

exchange ReRU in a peer-to-peer manner. The structure of this type in each server can be envisioned as shown in Figure 5.1.4.1.1. The header of this type contains the data items that are required for storing the instance information. The body is grouped according to two levels.

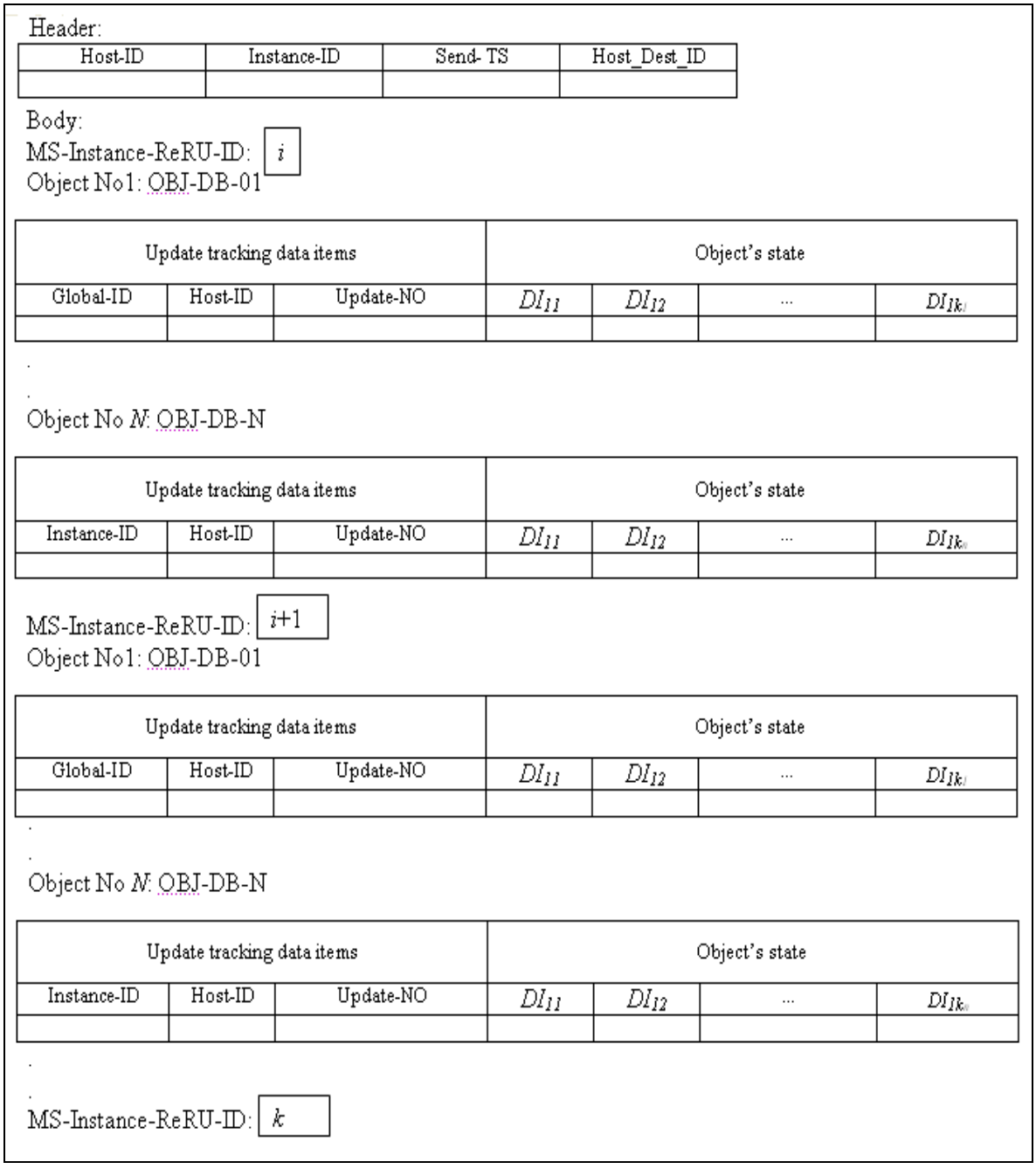


**Figure 5.1.4.1.1 The Description of ReRU-Propagation-VLDS**

The first grouping level is by the *Instance-ID* that migrated from the MS to underlying zone servers, which is called here *MS-Instance-ReRU-ID*. The value of *MS-Instance-ReRU-ID* is used to check last updates that are received from the higher level. The reason of grouping using this data item is that the host in the underlying level may not receive all the instances issued from the host in the higher level as previously mentioned. These instances are called missed instances. The second grouping level is by *Object-ID*.

In Figure 5.1.4.1.1, *i* is the number of the first missed instance out of *k* missed instances. In case of propagating *ReRU* from MS to ZS, the value of *Instance-ID* in the header of this type will be same as the *ID* of the last value of *MS-Instance-ReRU-ID* in the body, which here is *K*. The reason is that the number of instances that ZS missed will be up to *K*, which represents the number of last instance issued from MS. If the missed instance is the only the last instance, then the grouping in the body section will be only by the *ID* of the last instance. In case of propagating *ReRU* from ZS to C*S* or from CS *to* MH, the value of *Instance-ID* is serially assigned according to the value of *Instance-ID* in *Outgoing-Instances-Obj* (see appendix One) that is stored on the database of IIRA in the higher level. Similarly, as in propagating *ReRU* from MS to ZS*,* the values of *MS-Instance-ReRU-IDs* in the body will be according to the missed values of *MS-Instance-ReRU-ID* in the hosts in the lower level. The *Host-ID* in the header represents the *ID* of the instance's sender while the *Host-ID* in the body represents the *ID* of the host where update occurred.

#### *5.1.4.2 Propagating ReRU Algorithm*

The steps that are carried out in both hosts in the higher and lower levels for propagating ReRU are as follows. In these steps, the last instance that has been sent to the host in the lower level is denoted by *L-Sent*, while the last instance that should be propagated to the hosts in the lower level is denoted by *L-Last*.

***Step 1*** *(Checking the connection status with the host in the lower level)*

***If*** *the connection is established* ***then***

***If*** *Host-Type = 'MH'* ***then***

*1.1 MHR-IIRA sends a message to the CSR-IIRA includes the ID of the last instance received (MS-Instance-ReRU-ID in ReRU-Received-Instances-Obj).*

*Go to Step 3*

***Else***

*Go to Step 2*

***Else***

***If*** *Host-Type = 'MH'* ***then***

*1.2 Wait for connection with the fixed network.*

***Else***

*1.3 Check the connection status with another host in the lower level.*

***Step 2*** *The IIRA of the host in the higher level compares the ID of L-Sent (MS-Instance-ReRU-ID in Hosts object) with the value of the ID of L-Last (MS-Instance-ReRU-ID in ReRU-Received-Instances-Obj).*

***Step 3*** *(Checking the difference between ID of L-Sent and ID of L-Last)*

***If*** *Difference = 1* ***then***

*3.1 Selecting only the L-Last for propagation to the lower level.*

*3.2 Filling ReRU-Propagation-VLDS by selecting updates according to the ID of L-Last from each data object through linking each object with ReRU-Received-Instances-Obj object to fetch only those updates associated with that instance.*

***Else*** *(the difference >1)*

*3.3 Filling ReRU-Propagation-VLDS by selecting updates according to the IDs of the missed instances.*

*3.4 Grouping updates in the body according to these IDs.*

***Step 4*** *After filling ReRU-Propagation-VLDS, the instance migrates to the host in the lower level.*

*4.1 Storing the details of the instance in Outgoing- Instances-Obj object.*

***Step 5*** *(Execution of the instance on the lower level)*

*5.1 Inserting the updates that are shipped in ReRU-Propagation-VLDS in corresponding objects in the replicated database*

*5.2 Inserting the details of the instance in the ReRU-Received-Instances-Obj.*

***Step 6*** *(Finalizing the propagation task)*

*6.1 The instance sends an acknowledgment message to its parent IIRA in the sender indicates that the transferred updates are applied successfully.*

*6.2 The instance removes itself.*

***Step 7*** *The IIRA in the higher level removes the contains of the ReRU-Propagation-VLDS once the acknowledgment message is received.*

These steps are repeated by IIRA for each underlying host in order to ensure that the last ReRU is propagated to all available underlying hosts. In case of the host type is

master server, the object *ReRU-Received-Instances-Obj* is called *ReRU-Sent-Instances-Obj*.

In case of the host type is mobile host, the MHR-IIRA sends a message to the CSR-IIRA at the beginning of the connection period includes the *ID* of the last instance. This is because storing the *ID* of the last instance that is received by a mobile host in the hosts object in the same manner as in the case of zone and cell servers is not beneficial here, since the mobile host does not belong to specific cell or zone.

In step 7, The IIRA in the higher level removes the contains of the ReRU-Propagation-VLDS once the acknowledgment message is received from its all instances that migrated to the underlying hosts in case of propagating to multiple hosts (i.e. Mutli-cast propagation).

### *5.1.5 Implementing Peer-to-Peer Propagation*

The two IIRAs can exchange either ReRU or RU as follows.

**(i) Exchanging ReRU**

In this case, a message from each peer is sent to the other that includes the last value of *MS-Instance-ReRU-ID*, which is received by each of them. Depending on this value, each peer retrieves the updates that are associated with the instances, which the other peer missed and stores these updates in ReRU-Propagation-VLDS. Then, an instance from each peer holding ReRU-Propagation-VLDS will migrate to the other.

Exchanging ReRU is useful in case of two nearby MHs desire to get last ReRU and they cannot connect to the fixed network through any cell server (e.g. they may be far from the location of any cell server or unavailability of the nearby cell server). It is useful for cell servers and zone servers only in case of unavailability of the server in the higher level.

**(ii) Exchanging RU**

In this case, peers exchange their recent updates based on the value of the *Max-TS* data item, which is stored in the *Hosts-Replicated-Objects-Obj* object (see appendix One). The purpose of this data item is to store the maximum value of the *Update-TS* for updates in each object that are propagated previously to the other peer. Accordingly, the updates that

will be propagated in the next synchronization to that peer will have *Update-TS* values greater than the value of the *Max-TS* according to each object. For example, when $MH_a$ desires to synchronize with $MH_b$, each one of them propagates the updates that have *Update-TSs* greater than the value that is registered for the other peer in *Hosts-Replicated-Objects-Obj* for each object. Thus, there is no need for exchanging messages to avoid duplication in sending updates as in the first case.

### 5.2 Behavior Modeling

The research models the dynamic behavior of the proposed IIRA-based propagation system with respect to the synchronization process between its components by using SPNs. The purposes are to trace how the mobile database will reach the CA state and to calculate the steady-state probabilities.

The reason of using SPNs is that the research views the synchronization process between the different levels of the replication architecture as a discrete-event stochastic system that encompasses states and events. Each state captures either a snapshot of a set of recent updates or a snapshot of a set of tuples currently is stored in the database. Each event represents either execution of IIRA instance on another level or retrieving of a subset of tuples. The system is stochastic due to its stochastic state transitions, since it is evolving over continuous time and making state transitions when events associated with states occur.

The behavior modeling approach using SPN follows a systematic approach that is described in Balbo (2001) and Balbo (2007), which incorporates the following steps.

- Modeling of the behavior of the IIRA-Based propagation system using a stochastic Petri Net.
- Transforming the developed SPN into its equivalent Markov chain for calculation of the steady state probabilities of marking occurrences. This step requires generating the reachability graph. The Markov chain is obtained by assigning each arc with the rate of the corresponding transition.
- Analyze the Markov chain to obtain steady state probabilities.

The research interests in a state in which the mobile database in the mobile host receives a set of recent updates that occurred in the other hosts in both fixed and mobile networks.

### *5.2.1 SynchSPN*

The following definition will formally define the developed SPN that is used to model the behavior.

**Definition 5.2.1.1** SynchSPN is a six-tuple $<P, T, A, W, m_0, \Lambda>$ where:

1. $P = \{p_1, p_2, ..., p_{11}\}$ is a finite set of places, each place represents either a synchronization state for IIRA database (IIRA-Synch-State**)** or a synchronization state for the replicated database (DB-Synch-State**)** in each level. The former contains the set of recent updates, while the latter contains the set of tuples currently stored in the database.
2. $T= \{t_1, t_2, ..., t_{14}\}$ is a finite set of transitions. Each transition represents an operation carried out by the IIRA in different levels. These operations are:
   i. Retrieving the set of recent updates.
   ii. Execution of the IIRA instance on the other level to transfer the recent updates.
3. $A \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs that represents the number of updates, which have to be transferred after the execution process or the number of updates that have to be retrieved after the retrieving process.
4. $W: A \rightarrow \{1, 2, ...\}$ is the weight function attached to the arcs. This function maps each arc to the number of updates that are to be propagated in each synchronization process between the different levels.
5. $m_0: P \rightarrow \{0, 0, 0, 0, |MH\text{-}DBS|i, |CS\text{-}DBS|j, |ZS\text{-}DBS|k, |MS\text{-}DBS|, 0, 0, 0\}$ is the initial marking, which represents the number of recent updates at the synchronization state for each IIRA ($p_1, p_2, p_3, p_4, p_9, p_{10}, p_{11}$) and the number of tuples that are currently stored in each database ($p_5, p_6, p_7, p_8$). (In the context of discrete-event systems, the marking of the PN corresponds to the state of the system).
6. $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_{14}\}$ is the set of firing rates associated with the SPN transitions.

SynchSPN is developed based on the following assertion.

**Assertion 5.2.1.1** The synchronization process has Markov property.

**Proof.** Let $U(t)$ denotes the number of recent updates that should be propagated by IIRA instance from a host to another during their synchronization at time instant $t$, where $t$ varies over a parameter set $T$. The value of $U(t)$ is not deterministic because it depends on the number of generated updates, which means $U(t)$ is a random variable. Accordingly, the synchronization process can be defined as a family of random variables $\{U(t)|t\in T\}$, where the values of $U(t)$ represent the states of the synchronization process. Thus, the synchronization process represents a stochastic process. By introducing a flag data item to mark the updates that are propagated at the current synchronization instant $n$, we find that the number of the recent updates that should be propagated on the next instant $n+1$ equals to the number of unmarked updates at n, which represent the recent updates that occur after $n$. Therefore, the value of $U(n+1)$ depends only the value of $U(n)$ and not on any past states.

By using SPN to model the synchronization system (see Figure 5.2.1.1 and tables 5.2.1.1-5.2.1.3), the system is composed of eleven places and fourteen transitions.

**Table 5.2.1.1 Description of Places**

| Place | Description |
|---|---|
| $p_1$ | IIRA-Synch-State in $MH_i$ for execution in $CS_j$ |
| $p_2$ | IIRA-Synch-State in $CS_j$ for execution in $ZS_k$ |
| $p_3$ | IIRA-Synch-State in $ZS_k$ for execution in $MS$ |
| $p_4$ | IIRA-Synch-State in $MS$ for execution in $MS$ |
| $p_5$ | DB-Synch-State in $MH_i$ |
| $p_6$ | DB-Synch-State in $CS_j$ |
| $p_7$ | DB-Synch-State in $ZS_k$ |
| $p_8$ | DB-Synch-State in $MS$ |
| $p_9$ | IIRA-Synch-State in $CS_j$ for execution in $MH_i$ |
| $p_{10}$ | IIRA-Synch-State in $ZS_k$ for execution in $CS_j$ |
| $p_{11}$ | IIRA-Synch-State in $MS$ for execution in $ZS_k$ |

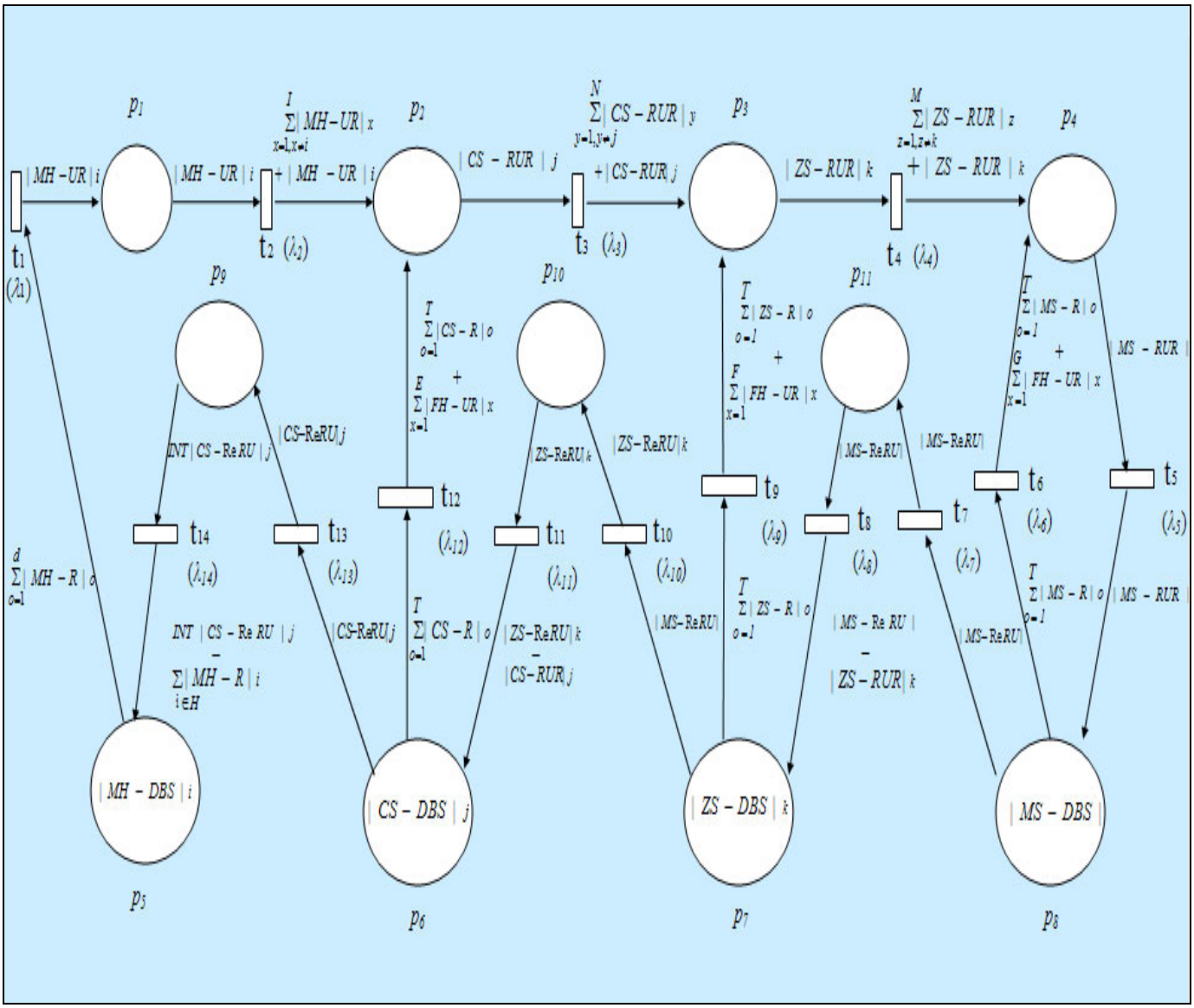


**Figure 5.2.1.1 SynchSPN**

### *5.2.1.1 Places*

They are called synchronization states. The research looks abstractly at the synchronization state as a state/place that contains a set of updates. There are three types of places:

- **IIRA-Synch-State for execution on the higher level:** It contains the set of recent updates that occurred in its host in addition to the set of collected updates from underlying level. For example, the synchronization state $p_2$ represents the set of all recent updates that are performed on cell server $j$ in addition to the set of all updates

that are transferred from mobile hosts, which are synchronized with this server in the last synchronization period. This type includes $p_1$, $p_2$, $p_3$, and $p_4$. Note that the set of all recent updates that occurred in the servers that exist in the fixed network are assumed that they include also the recent updates that are received from the fixed hosts in the level of those severs.

- **IIRA-Synch-State for execution on the lower level:** It contains the set of all recent resolved updates that are received from the higher level. For example, the synchronization state $p_{10}$ of zone server $k$ represents the set of all recent resolved updates that are received from the master server. This type includes $p_9$, $p_{10}$, and $p_{11}$.
- **DB-Synch-State:** This type stores the set of all tuples that are currently stored in the replicated database. It includes $p_5$, $p_6$, $p_7$, and $p_8$.

**Table 5.2.1.2 Description of Transitions**

| Transition | Description |
|---|---|
| $t_1$ | Retrieving recent updates from the replicated database in $MH_i$ |
| $t_2$ | Execution of MHR-IIRA instance on $CS_j$ |
| $t_3$ | Execution of CSR-IIRA instance on $ZS_k$ |
| $t_4$ | Execution of ZSR-IIRA instance on $MS$ |
| $t_5$ | Execution of MSR-IIRA instance on $MS$ |
| $t_6$ | Retrieving recent updates from the replicated database in $MS$ |
| $t_7$ | Retrieving recent resolved updates from the replicated database in $MS$ |
| $t_8$ | Execution of MSR-IIRA instance on $ZS_k$ |
| $t_9$ | Retrieving recent updates from the replicated database in $ZS_k$ |
| $t_{10}$ | Retrieving recent resolved updates from the replicated database in $ZS_k$ |
| $t_{11}$ | Execution of ZSR-IIRA instance on $CS_j$ |
| $t_{12}$ | Retrieving recent updates from the replicated database in $CS_j$ |
| $t_{13}$ | Retrieving recent resolved updates from the replicated database in $CS_j$ |
| $t_{14}$ | Execution of CSR-IIRA instance on $MH_i$ |

### *5.2.1.2 Transitions*

Also, the research looks abstractly at the transition as an event that leads to either retrieving or propagating the set of recent updates. There are three types of transitions: execution of the IIRA instance on the higher level, execution of the IIRA instance on the lower level, and retrieving of recent updates.

**Execution of the IIRA instance on the higher level.** This type inserts the contents of synchronization state for IIRA for execution on the higher level in the database of the IIRA type that inhabits the higher level. Some conditions must be satisfied in order to fire this type of transitions. These conditions are:

1. Enabling condition. It is composed of two conditions as follows.
    i. There is at least one recent update in the input place of the execution transition.
    ii. The connection with the other host should happen.

**Table 5.2.1.3 Notations and Their Meanings in SynchSPN**

| Symbol | Meaning |
|---|---|
| *MH*, *FH*, *CS*, *ZS, and MS* | Mobile Host, Fixed Host, Cell Server, Zone Server, and Master Server, respectively |
| *T* | Total number of database objects |
| *d* | Total number of database objects that are replicated in $MH_i$ (d<T) |
| $\|X\|$ | The number of updates in the set *X* |
| *MH-R*, *CS-R*, *ZS-R*, and *MS-R* | Set of recent updates for object *O* in $MH_i$, $CS_j$, $ZS_k$, and *MS*, respectively |
| *MH-UR*, *FH-UR* | Set of recent updates for all replicated objects in $MH_i$ and $FH_l$, respectively |
| *CS-RUR*, *ZS-RUR*, and *MS-RUR* | Set of resolved recent updates at $CS_j$, $ZS_k$, and *MS*, respectively |
| *MS-ReRU*, *ZS-ReRU*, and *CS-ReRU* | Set of recent resolved updates that are propagated to underlying level from *MS*, $ZS_k$, and $CS_j$, respectively |
| *MS-DBS*, *ZS-DBS*, *CS-DBS*, and *MH-DBS* | Replicated database state in *MS*, $ZS_k$, $CS_j$, and $MH_i$, respectively |
| *I* | Total number of mobile hosts that have synchronized with $CS_j$ before the synchronization of $MH_i$ during the $CS_j$ updates collection period |
| *N* | Total number of cell servers that have synchronized with $ZS_k$ before synchronization of $CS_j$ during the $ZS_k$ updates collection period |
| *M* | Total number of zone servers that have synchronized with *MS* before synchronization of $ZS_k$ during the *MS* updates collection period |
| *E, F, G* | Total number of fixed hosts that have synchronized with $CS_j$, $ZS_k$, and *MS*, respectively, during their updates collection period |
| *H* | Set of recent updates that are propagated from $MH_i$ to $CS_j$ in the last synchronization period |

The waiting time for the occurrence of the connection is not considered in the period that is required for firing transitions. This is because as previously mentioned, the waiting process for the connection is not considered as a required IIRA's operation for updates propagation. Moreover, the waiting time has a random value for mobile hosts and a deterministic value for the servers in the fixed network. Therefore, the research interests in the occurrence of the connection as a required condition for firing.

2. Completion of the updates conflicts resolution through ordering process for the collected updates.
3. Migration of the IIRA instance to the other host for execution of its parent synchronization state on that host.
4. Getting the permission for execution on the other host.

Each transition from this type can fire in a time instant equals to $T^{+}$ that is reached after elapsing of a time period of length $T^{up}$. The value of $T^{up}$ consists of the time period that is required for ordering the collected updates ($Ot^{+}$), the time period that is required for IIRA instance to migrate to the higher level host ($Mt^{+}$), and the time period that IIRA instance takes for waiting to get the permission for execution in the higher level host ($Wt^{+}$). Since, each one of these values is not deterministic; this means that $T^{up}$ is a random variable. Here, the time that is required for creating the instance is omitted, since it is performed locally.

**Execution of the IIRA instance on the lower level.** This type inserts the contents of synchronization state for IIRA for execution on the lower level into the replicated database of the host that inhabits the lower level. For firing this type, the same conditions that should be satisfied for the first type are applied here, excluding the completion of the updates ordering process.

Each transition from this type can fire in a time instant equals to $T^{-}$ that is reached after elapsing of a time period of length $T^{down}$. The value of $T^{down}$ consists of the time period that is required for IIRA instance to migrate to the lower level host ($Mt^{-}$) and the value of the period that IIRA instance takes for waiting to get the permission for

execution in the lower level host ($Wt^{r}$). Also, the time that is required for creating the instance is omitted, since it is performed locally in the same host.

**Retrieving recent updates.** This type involves either retrieving recent updates for synchronization with the higher level (propagating it to the higher level) or retrieving recent resolved updates for synchronization with the lower level. For firing this type, the enabling condition of the first type should be satisfied.

Each transition from this type can fire in a time instant equals to $T^{r}$ that is reached after elapsing of a time period of length $T^{ret}$. The value of $T^{ret}$ represents the time period that is required for IIRA to retrieve either recent updates or recent resolved updates from the replica. This value depends on the number of updates that should be retrieved in each synchronization process. Therefore, $T^{ret}$ is a random variable. The periods $T^{up}$, $T^{down}$, and $T^{ret}$ represent random variables because their values are obtained depending on non deterministic values.

Note that instead of adding a transition for representing the migration and its associated input place for representing the migrated state, they are incorporated into the execution transition and in the synchronization state. This is because the migrated state represents the synchronization state itself and the execution of the migration transition encompasses the synchronization state that already migrated to the other host. Thus, incorporation is performed to prevent the complexity of SynchSPN.

**Firing rates.** Each transition in SynchSPN is associated with a firing rate (i.e. the parameter λ). This is because the periods $T^{up}$, $T^{down}$, and $T^{ret}$ that represent the firing delays after correspondence transitions are enabled are random variables and are assumed exponentially distributed.

#### *5.2.1.3 The Initial Marking*

In this marking (i.e. $m_0$), the replicated databases that are stored on the servers in the fixed network (i.e. $CS_j$, $ZS_k$, and $MS$) are assumed identical. Also, the mobile database that is stored in the $MH_i$ is assumed that it has received a set of last updates. This received set may represent either all or a subset of the last updates, which occurred or propagated to the fixed network in the period that precedes the time of the last synchronization of $MH_i$ with the fixed network (i.e. during the disconnection time before the time of the last

synchronization), which equals to $MH_i\text{-}SynchT_{n-1}$ - $MH_i\text{-}SynchT_{n-2}$, where $MH_i\text{-}SynchT_{n-1}$ is the time of the last synchronization of $MH_i$ with the fixed network and $MH_i\text{-}SynchT_{n-2}$ is the time of the synchronization that precedes the last synchronization. Thus, according to the time of the last synchronization (i.e. $MH_i\text{-}SynchT_{n-1}$), the mobile database in $MH_i$ is assumed to be in CA state in the marking $m_0$ if it contains all recent resolved updates. And if for each marking $m^-$ reachable from $m_0$, the mobile database contains a set of recent updates, this means that $m^-$ is equivalent to $m_0$.

### *5.2.2 System Behavior*

The system behavior (i.e. evolution in time or dynamic changing of markings) is simulated by firing of transitions. The mechanism of firing in SynchSPN is based on the type of transition as follows.

- If *t* represents the execution of IIRA instance on the higher level, then *t* will remove the updates that exist in the input place and add them to the previously accumulated recent updates on the synchronization state of the IIRA in the higher level. The accumulated updates represent the updates received from other underlying hosts before the execution of IIRA instance on the higher level in the same time period for updates collection.
- If *t* represents the execution of IIRA instance on the lower level, then *t* will remove the recent resolved updates that exist in the input place and add them to the synchronization state of the replicated database of the host in the lower level and this happens after removing the set of updates that are propagated from the host in the lower level and are included in the set of the recent resolved updates. This is to avoid storing same updates once again.
- If *t* represents the retrieving of recent updates, then *t* will take a snapshot of the recent updates from the input place and add them to the synchronization state for IIRA for execution in either the lower or higher level. This transition does not remove the recent updates from the input source, which represent the synchronization state of the replicated database according to the fact that the retrieving process does not change the state of the database.

Note that when the connection takes place between any two hosts, the firing of the transition that represents the retrieving of recent updates always occurs before the firing of the other two types. This is because updates should be retrieved first before propagating them to the other host.

Based on the initial marking and the firing mechanism, the changing of the markings of SynchSPN is tracked starting from the time of the current synchronization of MHi with the fixed network, which is denoted by $MH_i$-$SynchT_n$ and ending with the time of the next synchronization, which is denoted by $MH_i$-$SynchT_{n+1}$. For obtaining a set of last updates, $MH_i$-$SynchT_{n+1}$ should occur in this tracking after firing of all transitions. The tracking of marking changing is also depends on the fact that the $MH_i$ will obtain the last updates that are performed on both fixed and mobile networks only after propagating these updates from their sources to the higher levels , where these updates are resolved. Therefore, bottom-up propagation is considered first then the top-down propagation. Thus, the firing sequence of the transitions is divided into two sequences as shown in Table 5.2.2.1**.**

**Table 5.2.2.1 The Firing Sequence of the Transitions**

| Propagation type | Firing sequence |
|---|---|
| Bottom-Up | $t_1 \rightarrow t_2 \rightarrow t_{12} \rightarrow t_3 \rightarrow t_9 \rightarrow t_4 \rightarrow t_6 \rightarrow t_5$ |
| Top-Down | $t_7 \rightarrow t_8 \rightarrow t_{10} \rightarrow t_{11} \rightarrow t_{13} \rightarrow t_{14}$ |

According to the specified firing sequence, the set of all reachable markings from $m_0$ are shown in Table 5.2.2.2. This set represents the evolution of the system in the period $MH_i$-$SynchT_{n+1}$ - $MH_i$-$SynchT_n$. The marking that reachable from firing of $t_{14}$ represents the marking in which the replicated database in $MH_i$ should obtain a set of recent updates that occurred or propagated to the fixed network during that period. This means that this marking is equivalent to $m_0$.

**Table 5.2.2.2 The Marking Table**

| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ | $p_9$ | $p_{10}$ | $p_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $m_0$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m^-_0$ | $\|MH\text{-}UR\|_i$ | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m_1$ | 0 | $T_{MH\text{-}CS}$ | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m^-_1$ | 0 | $\|CS\text{-}RUR\|_j$ | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m_2$ | 0 | 0 | $T_{CS\text{-}ZS}$ | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m^-_2$ | 0 | 0 | $\|ZS\text{-}RUR\|_k$ | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m_3$ | 0 | 0 | 0 | $T_{ZS\text{-}MS}$ | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m^-_3$ | 0 | 0 | 0 | $\|MS\text{-}RUR\|$ | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|$ | 0 | 0 | 0 |
| $m_4$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | 0 | 0 | 0 |
| $m^-_4$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | 0 | 0 | $\|MS\text{-}ReRU\|$ |
| $m_5$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k +$ $T_{MS\text{-}ZS}$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | 0 | 0 | 0 |
| $m^-_5$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j$ | $\|ZS\text{-}DBS\|_k +$ $T_{MS\text{-}ZS}$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | 0 | $\|ZS\text{-}ReRU\|_k$ | 0 |
| $m_6$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j +$ $T_{ZS\text{-}CS}$ | $\|ZS\text{-}DBS\|_k +$ $T_{MS\text{-}ZS}$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | 0 | 0 | 0 |
| $m^-_6$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i$ | $\|CS\text{-}DBS\|_j +$ $T_{ZS\text{-}CS}$ | $\|ZS\text{-}DBS\|_k +$ $T_{MS\text{-}ZS}$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | $\|CS\text{-}ReRU\|_j$ | 0 | 0 |
| $m_7$ | 0 | 0 | 0 | 0 | $\|MH\text{-}DBS\|_i +$ $T_{CS\text{-}MH}$ | $\|CS\text{-}DBS\|_j +$ $T_{ZS\text{-}CS}$ | $\|ZS\text{-}DBS\|_k +$ $T_{MS\text{-}ZS}$ | $\|MS\text{-}DBS\|+$ $\|MS\text{-}RUR\|$ | 0 | 0 | 0 |

In the marking table, the marking $m_7$ is equivalent to $m_0$ because the former represents the state in which the mobile database in MHi receives a set of recent updates that occurred or propagated to the fixed network during the period: $MH\text{-}Synch T_{n+1}$ - $MH\text{-}Synch T_n$

The marking table includes the following Equations:

$$T_{MH\text{-}CS} = \sum_{x=1, x\neq i}^{I} |MH - UR|_x + |MH\text{-}UR|_i \qquad (1)$$

Where $T_{MH\text{-}CS}$ is the total number of updates that will be propagated to *CSj* during its updates collection period from *I* mobile hosts.

$$|CS\text{-}RUR|_j = T_{MH\text{-}CS} + |CS\text{-}UR|_j \qquad (2)$$

This equation represents the total number of resolved updates that will be propagated from $CS_j$ to $ZS_k$.

$$T_{CS\text{-}ZS} = \sum_{y=1, y\neq j}^{N} |CS - RUR|_y + |CS\text{-}RUR|_j \quad (3)$$

Where $T_{CS\text{-}ZS}$ is the total number of updates, which will be propagated to *ZSk* during its updates collection period from *N* cell servers.

$$|ZS\text{-}RUR|_k = T_{CS\text{-}ZS} + |ZS\text{-}UR|_k \quad (4)$$

This equation represents the total number of resolved updates that will be propagated from $ZS_k$ to *MS*.

$$T_{ZS\text{-}MS} = \sum_{z=1, z\neq j}^{M} |ZS - RUR|_z + |ZS\text{-}RUR|_k \quad (5)$$

Where $T_{ZS\text{-}MS}$ is the total number of updates that will be propagated to *MS* during its updates collection period from *M* zone servers.

$$|MS\text{-}RUR| = T_{ZS\text{-}MS} + |MS\text{-}UR| \quad (6)$$

This equation represents the total number of resolved updates that will be stored in the database that exists in *MS*.

$$T_{MS\text{-}ZS} = |MS\text{-}ReRU| - |ZS\text{-}RUR|_k \quad (7)$$

Where $T_{MS\text{-}ZS}$ is the total number of resolved updates that will be propagated to $ZS_k$ database from *MS* excluding updates that previously propagated from $ZS_k$.

$$T_{ZS\text{-}CS} = |ZS\text{-}ReRU|_k - |CS\text{-}RUR|_j \quad (8)$$

Where $T_{ZS\text{-}CS}$ is the total number of resolved updates that will be propagated to $CS_j$ database from $ZS_k$ excluding updates that previously propagated from $CS_j$.

$$T_{CS\text{-}MH} = |CS\text{-}ReRU|_k - |MH\text{-}UR|_j \quad (9)$$

Where $T_{CS\text{-}MH}$ is the total number of resolved updates that will be propagated to MHi database from CSj excluding updates that previously propagated from MHi.

**Reachability graph.** This graph is described in Figure 5.2.2.1. The markings $m_i^-$ , where $i$=0,1,…,6 that reachable from firing of retrieve transitions are not included in the reachability graph because this type of transitions affects only the local synchronization state for IIRA database. However, these transitions are included, since their firing

precedes the firing of the execution transitions that leads to the markings $m_j$, where $j$=1, 2,…, 7.

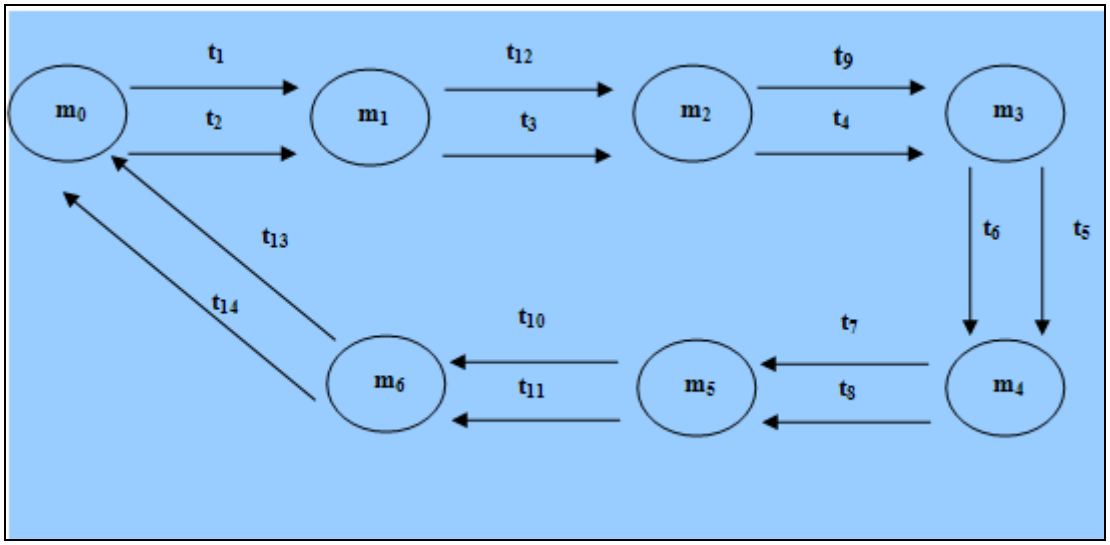


**Figure 5.2.2.1 Reachability Graph**

**Equivalent Markov chain.** In the derived CTMS (see Figure 5.2.2.2), the firing rates of the retrieve transitions are not considered because as mentioned previously, the retrieve operation is performed locally from the replicated database on a given host.

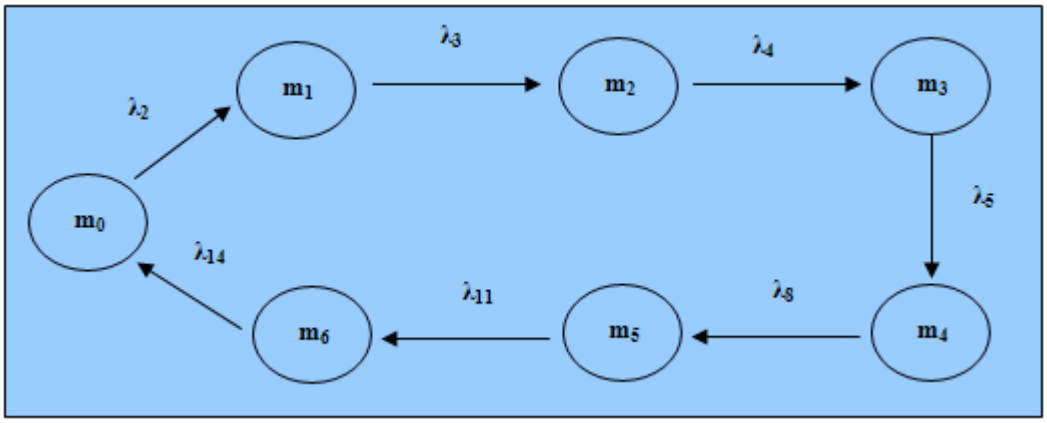


**Figure 5.2.2.2 Derived Markov Chain**

**Analysis of the Markov chain**. The steady-state probabilities, denoted by $\prod = (\pi_0, \pi_1, \pi_2, \ldots, \pi_6)$ are obtained by solving the following equations:

$$\prod A = 0 \qquad (10)$$

$$\sum_{i=0}^{6} \pi_i = 1 \qquad (11)$$

Where $A$ is the transition rate matrix. $\pi_i$ is the steady-state probability of marking that is equivalent to $m_i$.

The obtained matrix for the derived Markov chain is shown in Figure 5.2.2.3.

| | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ |
|---|---|---|---|---|---|---|---|
| $m_0$ | $-\lambda_2$ | $\lambda_2$ | 0 | 0 | 0 | 0 | 0 |
| $m_1$ | 0 | $-\lambda_3$ | $\lambda_3$ | 0 | 0 | 0 | 0 |
| $m_2$ | 0 | 0 | $-\lambda_4$ | $\lambda_4$ | 0 | 0 | 0 |
| $m_3$ | 0 | 0 | 0 | $-\lambda_5$ | $\lambda_5$ | 0 | 0 |
| $m_4$ | 0 | 0 | 0 | 0 | $-\lambda_8$ | $\lambda_8$ | 0 |
| $m_5$ | 0 | 0 | 0 | 0 | 0 | $-\lambda_{11}$ | $\lambda_{11}$ |
| $m_6$ | $\lambda_{14}$ | 0 | 0 | 0 | 0 | 0 | $-\lambda_{14}$ |

**Figure 5.2.2.3 Transition Rate Matrix**

By solving equation (10) and equation (11), the obtained steady-state probabilities as follows:

$$\Pi = \begin{pmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \\ \pi_5 \\ \pi_6 \end{pmatrix} = \begin{pmatrix} 1/(1+\lambda_2\omega_0) \\ 1/(1+\lambda_3\omega_1) \\ 1/(1+\lambda_4\omega_2) \\ 1/(1+\lambda_5\omega_3) \\ 1/(1+\lambda_8\omega_4) \\ 1/(1+\lambda_{11}\omega_5) \\ 1/(1+\lambda_{14}\omega_6) \end{pmatrix}$$

Where:

$\omega_0 = 1/\lambda_3+1/\lambda_4+1/\lambda_5+1/\lambda_8+1/\lambda_{11}+1/\lambda_{14}$

$\omega_1 = \omega_0-(1/\lambda_3+1/\lambda_2)$

$\omega_2 = \omega_0-(1/\lambda_4+1/\lambda_2)$

$\omega_3 = \omega_0-(1/\lambda_5+1/\lambda_2)$

$\omega_4 = \omega_0-(1/\lambda_8+1/\lambda_2)$

$\omega_5 = \omega_0-(1/\lambda_{11}+1/\lambda_2)$

$\omega_6 = \omega_0-(1/\lambda_{14}+1/\lambda_2)$

As previously mentioned, the research interests in the state in which the mobile database in the mobile host contains a set of recent updates. Therefore, the value of $\pi_0$ represents the probability of the marking that is equivalent to $m_0$.

**Assertion 5.2.2.1** There is only a subset $C \subseteq R$ ($m_0$), such that the mobile database in specific mobile host in CA State for each $m \in C$.

**Proof**. Let $t_j$ ($j=1,...,m$) denotes the transition that represents the execution state of the CSR-IIRA in $MH_i$, and $t_i$ ($i=1,...,n$) denotes the transition that represents the execution state of the MHR-IIRA in the fixed network. We show that firing of $t_j$ will result in CA State for $R$ that is hosted in $MHi \Leftrightarrow$ each $t_i$ is fired during the time period that precedes the current synchronization time of $MH_i$. Since the latter condition is not realized at all synchronization times for $MH_i$ due to existing of many MHs are not connected before the synchronization of MHi with the cell server. Therefore, the firing of $t_j$ leads to CA State $\Leftrightarrow$ all updates that are performed in the mobile network are propagated to the fixed network before the synchronization of $MH_i$. This means that if the latter condition is realized, the firing of $t_j$ will results in a marking that represents an element of $C$.

**Assertion 5.2.2.2** The probability that the mobile database in CA state is:

$$P(\text{CA}) = n\pi_0 \qquad (12)$$

Where $n$ is the number of synchronization times for $MH_i$ with the fixed network that led to the CA state.

**Proof.** Let $C$ be the subset of $R(m_0)$ satisfying the condition that the place $p_5$ has received all recent updates that occurred and resolved before the synchronization time of $MH_i$ with the fixed network, which led to each marking in $C$. Then the probability of this condition is: $P(C) = \sum_{i \in C} \pi_i$ , Since the probability that $MH_i$ receives a set of recent updates is $\pi_0$. Then $\pi_i = \pi_0$ for each marking $m_i \in$ C. This means that $P(C) = \sum_{i=1}^{n} \pi_i = n\pi_0$ , where $n$ is the number of markings in $C$. This number is equivalent to the number of the synchronization times with the fixed network that led to storing all recent updates in $p_5$.

**Assertion 5.2.2.3** The marking $m_0$, where $p_5$ receives a set of recent resolved updates is recurrent after a time period equals to: $MH_i\text{-}SynchT_{n+1} - MHi\text{-}SynchT_n$ , where $MHi\text{-}SynchT_{n+1} - MHi\text{-}SynchT_n > \lambda_5 + \lambda_8 + \lambda_{11}$ , $MHi\text{-}SynchT_n$ is the time instant of the current synchronization with the fixed network, $MH_i\text{-}SynchT_{n+1}$ is the time instant of the next synchronization with the fixed network.

**Proof.** $m_0$ is recurrent if the following equation is satisfied.

$$\sum_{i=MHi\text{-}SynchTn}^{MHi\text{-}SynchTn+1} P^{i}_{m_0 m_0} = 1 \qquad (13)$$

Where $P^{i}_{m0m0}$ is the probability that the system returns to state $m_0$ starting from $m_0$. Suppose that the markings that are reachable from $m_0$ occur at the following time instants:

$MHi\text{-}SynchT_n, T_1, T_2, \ldots, T_n, MHi\text{-}SynchT_{n+1}$, where $MHi\text{-}SynchT_n < T_1 < T_2 < \ldots < T_n < MHi\text{-}SynchT_{n+1}$ . Obviously, the $MH_i$ can obtain the last updates that occurred or are propagated to the fixed network during the period: $MHi\text{-}SynchT_{n+1} - MHi\text{-}SynchT_n$ after a time instant $> = MHi\text{-}SynchT_{n+1}$, which means that:

$P^{MHi\text{-}SynchTn}{}_{m0m0} = 0$, $P^{MHi\text{-}SynchTn+1}{}_{m0m0} = 1$, $P^{T1}{}_{m0m0} = 0$, $P^{T2}{}_{m0m0} = 0$, …, $P^{T0}{}_{m0m0} = 0$

To obtain the latest updates, the following condition should hold:

$$MH_i\text{-}SynchT_{n+1} - MH_i\text{-}SynchT_n > \lambda_3 + (\lambda_4 - \lambda_3) + (\lambda_5 - \lambda_4) + \lambda_8 + \lambda_{11} = \lambda_5 + \lambda_8 + \lambda_{11}$$

Where $\lambda_3 < \lambda_4 < \lambda_5$. This is because the server in the master level receives updates from all underlying levels, while the servers in the zone and cell levels receive updates from the hosts that are located in their areas.

**Assertion 5.2.2.4** The SynchSPN is deadlock free.

**Proof**. We prove that $\forall\ m \in R\ (m_0)$, $\exists\ m^- \in R\ (m)$ such that $\exists\ t$ is enabled for $m^-$, where $R\ (m_0)$ is the set of markings reachable from $m_0$ by firing a sequence of transitions. Recall that our assumptions regarding that tens of updates per each data item are expected at any period of time and the connection is reliable and fixed between the servers of the fixed network, this means that the following conditions are true:

1. At any period of time, $\exists\ DB \in RDB$ such that $DB$ is updated recently (has UR) where $RDB$ is the set of the replicated databases in either fixed or mobile hosts. This condition ensures enabling of selection transactions.
2. There is at least one host (either fixed or mobile) is synchronized with other host (e.g. MH with a cell server or other MH, FH with a server, CS with ZS…etc). This condition ensures that there is at least one of the execution transitions is enabled after satisfying condition one.

## 5.3 Summary

This chapter described IIRA-based propagation system that is proposed for implementing automated updates propagation between the components of the replication system. The proposed propagation system consists of the four types of IIRA. Updates are exchanged between these types via their created instances when the connection occurs (in the case of mobile hosts) or when the updates collection period elapses (in the case of the servers).

SPN is developed to model and analyze the dynamic behavior of the replication system in regard to reaching the Consistent-Available state for the replicated database at the mobile host. The analysis proved that the Consistent-Available state for mobile database is a recurrent state with a probability that depends on the number of the synchronizations of a mobile host with the fixed network. Accordingly, bounding of

unavailability of recent updates and inconsistency of stored data depends on the number of connections with the fixed network. Also, analysis explored that IIRA-based propagation system achieves load balance in both updates propagation and ordering processes.