DOWNTOWN LINE

Contract C955

InteGRATED SUPERVISORY

CONTROL SYSTEM (iscS)

distributed heterogeneous database Synchronization design spec

**AMENDMENT RECORDS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Rev*** | ***Date*** | ***Descriptions*** | ***Writer*** | ***Function*** |
| *1* | *17/04/2012* | First draft | Ouyang Zhilin | Module design note |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Reviewed by** |  | **Approval by** |  |
| **Entity / Function** |  | **Entity / Function** |  |
| **Date & Sign.** |  | **Date & Sign.** |  |
| **Reserved to the Engineer** |  |  |  |

**Table of Contents**

[1 distributed heterogeneous database replication design 5](#_Toc330539655)

[1.1 General 5](#_Toc330539656)

[1.1.1 Purpose 5](#_Toc330539657)

[1.1.2 Version Control 5](#_Toc330539658)

[1.1.3 Definitions,Acronyms and Abbreviations 6](#_Toc330539659)

[1.1.4 References 6](#_Toc330539660)

[2 Motivation 6](#_Toc330539661)

[2.1 Background 6](#_Toc330539662)

[2.2 Requirements 6](#_Toc330539663)

[2.2.1 Formal Requirements 6](#_Toc330539664)

[2.2.2 Derived Requirements 9](#_Toc330539665)

[3 GAP Analysis between C955 and C830 9](#_Toc330539666)

[3.1 Configuration Data Replication 9](#_Toc330539667)

[3.1.1 Gap solution 10](#_Toc330539668)

[3.2 Audit Data Replication 10](#_Toc330539669)

[3.2.1 Gap Solution 10](#_Toc330539670)

[3.3 MMS request data transference 11](#_Toc330539671)

[3.3.1 Gap Solution 11](#_Toc330539672)

[3.4 Station database clustering 11](#_Toc330539673)

[3.4.1 Gap Solution 11](#_Toc330539674)

[4 solution 11](#_Toc330539675)

[4.1 Assumptions and Constraints 11](#_Toc330539676)

[4.1.1 System Environmental Constaints 11](#_Toc330539677)

[4.2 Major Design Decisions 12](#_Toc330539678)

[4.2.1 System Decomposition Decisions 12](#_Toc330539679)

[4.2.1.1 Data Capture Module 12](#_Toc330539680)

[4.2.1.2 Data Queue Module 12](#_Toc330539681)

[4.2.1.3 Data Propagation Module 13](#_Toc330539682)

[4.2.1.4 Data Reconstruction Module 13](#_Toc330539683)

[4.2.2 Threading and Concurrence Design Decision 13](#_Toc330539684)

[4.2.3 Data Consistency&Integrity Design Decision 13](#_Toc330539685)

[4.2.4 MMS Agent Message Design Decision 14](#_Toc330539686)

[4.3 Related cots 14](#_Toc330539687)

[4.4 Detail design 14](#_Toc330539688)

[4.4.1 Data Capture Module 14](#_Toc330539689)

[4.4.2 Data Queue Module 14](#_Toc330539690)

[4.4.3 Data Propagation Module 14](#_Toc330539691)

[4.4.3.1 Simplified two-phase commit protocol 14](#_Toc330539692)

[4.4.3.2 Propagation clustering solution 15](#_Toc330539693)

[4.4.3.2.1 Central 16](#_Toc330539694)

[4.4.3.2.2 Station 17](#_Toc330539695)

[4.4.3.2.3 Upload 17](#_Toc330539696)

[4.4.3.2.4 Download 17](#_Toc330539697)

[4.4.3.3 Network partition handling 17](#_Toc330539698)

[4.4.3.3.1 Group proxy connection Strategy 19](#_Toc330539699)

[4.4.3.3.2 Message Enqueu Strategy 20](#_Toc330539700)

[4.4.3.3.3 Download Control Strategy 21](#_Toc330539701)

[4.4.4 Data Reconstruction Module 22](#_Toc330539702)

[4.4.5 Class diagram 22](#_Toc330539703)

[4.4.6 Sequence diagram 22](#_Toc330539704)

[4.4.7 State diagram(if necessary) 25](#_Toc330539705)

[4.4.8 Component diagram 26](#_Toc330539706)

[4.4.9 Degrade mode 28](#_Toc330539707)

[4.4.10 Design capacities 28](#_Toc330539708)

[4.4.11 Threading and concurrence 28](#_Toc330539709)

[4.4.12 Error Handling 29](#_Toc330539710)

[4.4.13 Failover 29](#_Toc330539711)

[4.4.14 Interface 33](#_Toc330539712)

[4.4.15 Message 33](#_Toc330539713)

[4.4.16 Dependencies 36](#_Toc330539714)

[4.4.17 Access Control 37](#_Toc330539715)

[4.4.18 Configuration Information 37](#_Toc330539716)

[4.4.19 System Controller Group off-line Handling 38](#_Toc330539717)

[4.4.20 Database Design Changes/Extensions 39](#_Toc330539718)

[4.5 Safety function capability 40](#_Toc330539719)

[4.6 Testability 40](#_Toc330539720)

[4.7 Maintainability 40](#_Toc330539721)

[4.8 Impact Analysis 40](#_Toc330539722)

[4.9 Deployment considerations 40](#_Toc330539723)

[4.10 Flow-on Effects 41](#_Toc330539724)

[4.11 Extensibility and Reuse 41](#_Toc330539725)

[5 gui-component design information 41](#_Toc330539726)

[Appendix A: Software module design review 41](#_Toc330539727)

# distributed heterogeneous database replication design

## General

### Purpose

This document has been written as an overall architecture of the middle-ware for the Distributed Heterogeneous Database Synchronization. It provides an overview of the middle-ware structure and the strategy to synchronize data between heterogeneous database servers (currently support Oracle and MySQL). This document also makes reference to the Distributed\_Database\_Architecture\_Requirements\_Note.doc & Database\_Structure\_Note.doc

### Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes / Reason for Issue |
| 0.01 | 16 Apri 2011 | Ouyang.Zhilin | First Draft |
| 1.00 |  |  | Issued for Release |
|  |  |  |  |

### Definitions,Acronyms and Abbreviations

Refer to the System Glossary **Error! Reference source not found.** and Project Glossary **Error! Reference source not found.** for definitions, acronyms and abbreviations.

### References

|  |  |  |
| --- | --- | --- |
|  | Reference | Title |
|  | [9999-P01-01-0001](I:\\STE\\Doc\\Project Documents\\C955_docs\\Database\\DbSynchDesign\\P-Product\\P01-Common\\P0101-General\\9999P01010001-System_Glossary.doc) | System Glossary |
|  |  | Software requirement |
|  |  | Software ICD |
|  |  | Software Architecture Specification |
|  |  | IEC62279 Phases |

# Motivation

This section describes the motivation of the DB SYNCH ENGINE including the background and major requirements of the sub-system.

## Background

Due to the differences of the Database Deployment between project C955 and C830, the data replication mechnism need to revise based on the new requirements.

In the existing project C830 both the local and central databases are Oracle, and therefore the data replication mechanism used Oracle built-in AQ function. The new project C955, however, the type of database servers will be defined as below:

* Central Database – residing at a central location accessible across a WAN. For C955 this is the OCCDB using Sun Oracle Enterprise database server.
* Local Database – associated with any physical location accessible from other local servers and central server across a WAN. For C955 this database will use MySQL database and will be considered as the Local Database for agents running on the location for the purpose of this design.

Based on the above design, data replication is required to support the distributed heterogeneous database architecture. Solutions & techniques needed to solve the replication between the Centre DB and Local DB.

## Requirements

This section is to outline the requirements for the distributed heterogeneous database synchronization that can be use as a basis for developing possible solution using appropriate technologies.

### Formal Requirements

The distributed heterogeneous database provides data redundancy and addresses performance requirements for reading from and writing to heterogeneous database.

Data stored in database can be categorized by data type for the purposes of distribution and access. This is to ensure that the data is stored appropriately in order to provide the necessary redundancy and performance. Currently the data types can be categorized as follows:

* CD – Configuration Data
* AD – Audit Data

Whilst data is categorized as above, not all grouped data have the same behavior with respect to the application requirements.

* Configuration Data

For the Configuration Data (CD), it’s very important to notice that the strategy for the modification of configuration data is along the following guidelines.

*Configuration Data shall be modified at the central DB server only. Where possible, Configuration Data from the central DB server shall be replicated to a local server where it is accessed for reading purpose.*

The purpose of these guidelines is to achieve the objectives for which distributed heterogeneous database architecture is designed. That is, to

* Provide data redundancy in case of a local server becoming isolated;
* Alleviate load from a central database server;
* Alleviate network traffic congestion.

The distribution method which meets the above objectives listed below:

**CD data flow strategy --- CD read from centre database with notification**

Configuration data is updated at the central database by Agents, Managers, and Configuration Editors. The data is automatically replicated to the Local database servers at periodic intervals. Configuration data changes for online updatable data initiates a broadcast notification and the data is reloaded from the central database.

When a Local server becomes isolated, the configuration data is read locally. Optional some writes may occur locally, but these writes will be temporary only and will be overwritten when the Central Database replication resumes.



Figure: CD data flow strategy

Configuration Data will not be copied to local servers by subset. That is, copying local site relevant configuration data only will not be attempted. The entire set of central controlled configuration data will be replicated to all local sites.

For the purposes of replication, related database tables must belong to the same refresh group to maintain database integrity.

* Audit Data

The general strategy for audit data is that it is written to and read from a local database server. Any audit data that is required immediately at the central server shall be immediately propagated to the central database server. Not all Audit Data is required to be replicated to the central database server.

**AD data flow strategy --- Local DB writes with replication to centre DB**

*The following figure* illustrates a category of data flow and data access for Audit Data in a distributed heterogeneous database architecture.



Figure: AD data flow strategy

All the audit data shall be read from and write to at a local database, for the purpose of replication, these data shall be periodically propagated to the central server database.

### Derived Requirements

# GAP Analysis between C955 and C830

This section describes the gap between project C955 and C830 and conducts the alternal solution for the gap. As the type and deployment of database server being changed for project C955, the data replication need new strategies and mechnism to achieve the same results and behaviors as the existing one by using Oralce built-in funcationalities like Materialized View (MW), Advance Queue (AQ).

## Configuration Data Replication

C830

The configuration data replication & redundancy used the built-in Oracle MW to synchronize the data periodically, as all the database servers are Oracle.

C955

The type of database server are change to local (MySQL) and Central (Oracle), therefore a new data replication mechanism need to implement to achieve the same behavior/results as the existing one does.

### Gap solution

For the configuration data replication, the following alternal solutions can be used.

* Use manual deployment to setup & config the configuration data.
* Implement middle-ware component to achieve the same results like the Oracle MV mechanism.

## Audit Data Replication

C830

The audit data replication & redundancy used the following mechanism to achieve data synchronization between database servers automatically.

*DAI + store procedure package + Oracle Advance Queue + Oracle AQ propagation*

C955

As the type of the local database server change to MySQL, a new mechanism & strategy need to implement to synchronize audit data between these distributed heterogeneous database systems automatically.

### Gap Solution

In order to imitate and achieve the same behavior like the existing one, an AQ mechanism in MYSQL needs to be implemented, a new propagation mechanism between Oracle and MYSQL needs to be introduced.

The solution is illustrated below.

● High availability

● Data consistence

● Data integrity

● Performance

● loading and bandwidth control

● Security

Dequeue

MYSQL Queue

Enqueue

Producer

Consumer

Oracle Queue

MYSQL Queue

Propagation

Propagation

Dequeue

Dequeue

Consumer

Consumer

## SynchEngine Monitor Controller

C830

Oracle AQ and with the assistance of schedule job in charge of all the message propagate flow and handle failover mechanism.

C955

As heterogeneous database introduced, a new monitor controller & failover strategy need to implement to monitor the SynchEngine for propagating message between heterogeneous database servers.

### Gap Solution

For the monitor of the SynchEngine both at MySQL & Oracle DB servers, the following solution can be considered:

* At Station DB server (usually is MySQL DB)

1. The SynchEngine as a plug-in for MySQL server, and the lifetime of it with MySQL DB server.
2. Based on the existing process(s) controller mechanism, the new introduced SynchEngine can work with the existing SystemController to monitor its status and handle failover mechanism.
3. As the SynchEngine only running under Sun Solaris based on our currently design, the cron job of Solaris can be used to monitor the process based on the MySQL DB server periodically.
4. Write another control/monitor process like SystemController to monitor SynchEngine and handle failover strategy.

* At Center(DBS) DB server (Enterprise Oracle DB)

1. The SynchEngine as a plug-in for Oracle server, and the lifetime of it with Oracle DB server.
2. As the SynchEngine only running under Sun Solaris based on our currently design, the cron job of Solaris can be used to monitor the process based on the Oracle DB server periodically.
3. Move the existing SystemController to DBS server and monitor the SynchEngine as station.
4. Write another control/monitor process like SystemController to monitor SynchEngine and handle failover strategy.

* Final Decision:

1. Station:

MySQL database: using combination of item 1 & 3. It based on the database replication status and combined with the shell script cron job of the OS.

Oracle database: as the propagation is new and independent from the existing project, using item 3.

1. Center (DBS):

Oracle database: as the propagation is new and independent from the existing project , using item 2.

## Station database clustering

C830

In C830 project, each station was equipped with a database server. The server runs a single database instance, there are no database clustering.

C955

Each station is equipped with two database servers. The two database servers are grouped as database clustering.

### Gap Solution

Station database clustering should be transparent with the distributed heterogeneous database replication.

# solution

## Assumptions and Constraints

This section lists the assumptions and constraints of the replication solution.

### System Environmental Constaints

* This middle-ware should be working properly both in Sun Solaris and Ms Windows platforms.
* This middle-ware will be working in a distributed network based system with low/high bandwidth speed through WAN cable.
* This middle-ware should communicate and synchronize data properly through heterogeneous database servers in highly performance with heavy load data traffic.

## Major Design Decisions

### System Decomposition Decisions

This section provides the decomposition structure of the SynchEngine system for synchronizing data between distributed heterogeneous database servers. Five modules were introduced to facilitate and meet the functional requirements. The purpose of each module describes below:

#### Data Capture Module

To capture the data change/modify events of the database. Work collaborates with DAI and database procedure to en-queue the message into queue table based on SQL statement. The sequence as below:

1. Agents invoke DAI.
2. DAI invoke AQ procedures. The routine of procedure should be careful design at MySQL side, as it easy cause deadlock.
3. Apply the SQL statement to local database.
4. Enq to queue tables.

#### Data Queue Module

In order to protect data loss after capturing data, captured data need to be persisted.

Oracle AQ is used in oracle database, an AQ mechansim, named as MYSQL AQ, needs to be introduced in MYSQL database. The MYSQL AQ should at least provide the following basic funcationality.

1. Enqueue function

Supportting raw data and user defined data

Allowing to enqueue a message that is targeted to a specific set of consumers.

Different data can be put into different queue.

Supporting local or remote queue propagation.

1. Dequeue function

Data can be dequeued by FIFO + Priority order or based on a specific message ID.

Mode of Dqueueing:

Supporting to view a message in “brower” mode.

Supporting to delete a message either the “remove” mode or “remove with no data” mode.

Supporting dequeue transaction

Supportting “pull” mode of message delivery.

Exceptin handling

AQ provides four integrated mechanisms to support exception handling in applications: EXCEPTION\_QUEUES, EXPIRATION, MAX\_RETRIES and RETRY\_DELAY.

1. Queue Management

Adding Queue

Removing Queue

Modifing Queue

#### Data Propagation Module

All change operations to the data is based on SQL statements such as Insert/Delete/Update etc. Therefore this sub-system is based on SQL statements to synchronize data between heterogeneous database servers.

All propagated message with FIFO order should be indeed received at the remote end ,and received only once.

The data propagation main sequence as below:

1. Dequeue data from Queue
2. Propagating the data from local queue to remote queue based on enqueue request.
3. If the step 2 runs succesfully, remove the data from local queue queue.
4. If the step 2 failed and the failure reason is lossing network connection, the step 2 will be tried again after a predefined time interval passed.
5. If the step 2 failed and the failure reason is that remoted database down or failure, the step 2 will be tried again after a predefined time interval passed.
6. If the step 2 failed and failure reason is that the data cannot be inserted into remote queue, the data will be insert into remote exception queue, then goes to step 1 and starts a new progation process.
7. If the remote exception is failed, how to do.
8. If still failed, how to handle the data.

#### Data Reconstruction Module

After propagating data to queue of remote database, the data need to be dequeued from queue of remote database and apply these SQLs into remote database. Its flow as below:

1. Dequeue data from remote queue.
2. Apply the dequeue data into remote database.

The above modules will work cooperate with each other and finish the synchronization mechanism between the distributed heterogeneous database servers. The following diagram details one way of data transferring.

Source

DB

Data Capture

Target

DB

Data Queue(MYSQL/Oracle)

Data Propagation

Data Queue(MYSQL/Oracle)

Data Reconstruction

Figure: Data Synchronization Solution

### Data Capture Module

1. Both MYSQL SQL and Oracle SQL for an agent’s SQL need to be capture, and they need to be propagated within heterogeneous database.
2. The store procedure which is used for submitting SQL by agents will be kept in oracle database, and create a store procedure with same functionality in MYSQL database.
3. An enqueue store procedure needs to be provided in MYSQL database.

### Data Queue Module

For the detail design, please refer to MySQL\_Database\_AQ\_Library\_Design\_spec.docx

### Data Propagation Module

For the detail design, please refer to Distributed\_Database\_Propagation\_Design\_spec.docx

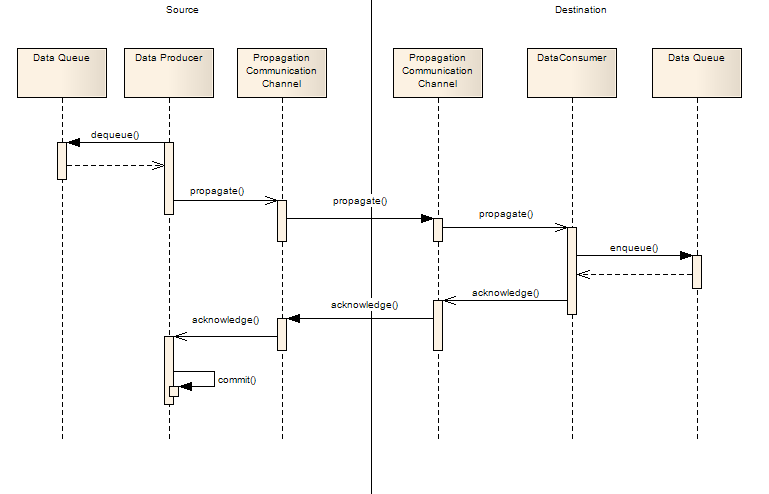
#### Simplified two-phase commit protocol

A simplified two-phase commit protocol is applied in the module. Due to the propagation only involved Data Producer (act as coordinator) and Data Consumer (cohort), the “Commit-request” phase has to be moved. The following steps describe the main happy flow:

1. Data Producer dequeue data with FIFO order from Data Queue
2. Data Producer launches data propagation and waits acknowledge response.
3. Propagation communication channel of source transfers data to propagation communication channel of destination.
4. Propagation communication channel of destination propagates data to data consumer.
5. Data consumer enqueue data into Data Queue
6. Data consumer send an acknowledge response to data producer via propagation channel
7. Data producer gets the acknowledge response from propagation channel.
8. Data producer commits the dequeue operation and go to step 1 for starting a new propagation transaction.

The protocol is illustrated below:

.



As the above graphic shows that data producer needs to get acknowledge response from data consumer, then data producer determines whether it should go to next propagation operation or retry current propagation operation. The following scenarios needs to be addressed:

1. Data should be only enqueu once;

Data in queue has an unique ID, it makes sure that no duplicated data exist in data queue; but if other process dequeue data, it may cause that duplicated data are enqueued.

1. Data consumer get data from propagation communication channel , but routine in the data consumer hang up.
2. Data consumer cannot enqueue data into its data queue.
3. Data Consumer enqueue successfully, but the acknowledge message is failed to send to data producer.
4. Data producer is crashed and failed to commit data

#### Propagation clustering solution

In order to provide high availability propagation service, a propagation clustering solution needs to be implementted. The solution is illstruated below.

**Upload**

**Download SMS01**

SMS DB01

Propagation Proxy1

Propagation Proxy2

SMS DB02

SMS0101

SMS DB01

Propagation Proxy1

Propagation Proxy2

SMS DB02

SMS0202

DBS DB

Propagation Proxy1

……….

Propagation ProxyN

DBS

**Upload**

**Station**

**Download SMS01**

Propagation Clustering Diagram

In order to propagate data among stations and CMS DB, propagation proxy is introduced (also named as SyncEngine).

Propagation proxy is responsible for propagating data from Station to Central (Upload) and Central to Station (download).

The above diagram shows that:

##### Central

In order to balance loading in CMS server, CMS needs to be deployed a group of propagation proxies; propagation proxy in CMS (CMS proxy) needs to connect with station propagation proxy (station proxy), there are two connection methods;

* Clustering group solution
* The other is that CMS proxy initiates connection;

CMS proxy need to be configured with which station proxies need to connect, and Active-Standby mechanism also need to be provide.

* One is that station proxy initiate connection to CMS proxy,

It does not require Active-Standby mechanism; any instance of CMS proxy can accept connection from any station proxy. It is necessary to provide a connection management mechanism to make sure that a CMS proxy serves reasonable station proxy. If all CMS proxies die, propagation will be stopped; a process monitor mechanism needs to be introduced for fixing the problem.

##### Station

As the above diagram shows that there are two application servers in each station to provide high availability services. As the same reason, propagation proxy is required to be deployed in every station application server. Station proxy works as an agent and is managed by system controller.

##### Upload

During normal mode, only control mode station proxy can upload data to CMS proxy (or group station proxy). Message Enqueue Strategy is used to prevent duplicated data.

##### Download

During normal mode, only control mode station proxy can accept download data from CMS proxy (or group station proxy). Download Control Strategy is used to prevent duplicated download.

#### Network partition handling

If two station application servers are in network partition status, then both of two station proxy runs as control mode. In our system, station network broken or brain leak is a crucial accident. All componments which resides at both server nodes will be isolated, and may also cause data inconsistence, replication between two server nodes failure. Therefore, manual assistance possible needs to involve solving this issue.

There are two kinds of senarios which will cause SynchEngine both running Active statuses, such as:

* SystemController heartbeat failed.
* Local network issue causes SystemController heartbeat failed.

The below table elaborates all possible scenarios.

|  |  |  |
| --- | --- | --- |
| Scenario | Propagation path | Sub scenario and solution |
| System Control Failure  Database good  DB Replication good  Network good | Upload  (station to center) | 1. Station cannot connect to central   Both SynchEngines retry to connect to central until connection is established. If connection is recovery, go to scenario 2 or scenario 3.   1. Both SynchEngines connect to Central   Central side is stateless. Both SynchEngines can upload data from station to central, central uses **Message Enqueue Strategy** to prevent duplicated data.   1. One of SynchEngine connects to Central   It is a special flow of the scenario 2. |
| Download  (central to station) | 1. Central cannot connect to station   Both SynchEngines retry to connect to central until connection is established. If connection is recovery, go to scenario 2 or scenario 3.   1. Central can connect to both SynchEngine   Because central side is stateless, central only allows creating one download channel from central to one station. Central uses **Download Control Strategy** to prevent duplicated download. In station side, **Message Enqueue Strategy** also need to be used.   1. Central interact with one of the SynchEngine   It is a special flow of the scenario 2 |
| Upload  (Station to group station) | 1. Only source system control failure   Same as station to central (upload data)   1. Both system controls failure   **Message Enqueue Strategy** is used to prevent duplicated data and manual assistance possible needs to involve solving this issue |
| Download  ( group station to station) | 1. Only source system control failure   Same as station to central (download data)   1. Both system controls failure   **Message Enqueue Strategy** is used to prevent duplicated data and **Download Control Strategy** is used to prevent duplicated download. |
| System Control Failure Isolated Station Database Network good  Or  System Contro Failure Network Failure  Isolated Station Database | Station to central (Upload data) | 1. Station cannot connect to central   Both SynchEngines retry to connect with central until connection is established. If connection is recovery, go to scenario 2 or scenario 3.   1. Both SynchEngines connect to Central   Central side is stateless. Both SynchEngines can upload data from station to central. Because two station servers create data independently, SynchEngine needs to upload data created on both station’s servers. To simplify the upload logical, system identify data using its input database. **Message Enqueue Strategy** is used to prevent duplicated data.   1. One of the SynchEngines connects to Central   It is a special flow of the scenario 2. |
| Central to station (Download) | 1. Central cannot connect to station   Both SynchEngine retry to connect to central until connection is established. If connection is recovery, go to scenario 2 or scenario 3.   1. Central can connect to both active SynchEngine   Central uses **Download Control Strategy** to prevent duplicated download.; If central switchs between these two SynchEngines, a **mannul merging strategy** is required.   1. Central can connect to one of SynchEngine   It is a special flow of the scenario 2. |
| Upload  (Station to group station) | 1. Only Source failure   **Message Enqueue Strategy** is used to prevent duplicated data.  If two source application server create business involved data, a **mannul merging strategy** is required while the source recovery   1. Both failed   **Message Enqueue Strategy** is used to prevent duplicated data. But both SynchEngines connect to group member respectively. a **mannul merging strategy** is required. |
| Download  ( group station to station) | 1. Only Source failure   **Message Enqueue Strategy** is used to prevent duplicated data.  If two source application server create business involved data, a **mannul merging strategy** is required while the source recovery   1. Both failed   **Message Enqueue Strategy** is used to prevent duplicated data. But both SynchEngines connect to group member respectively. a **mannul merging strategy** is required. |

Table: SystemController heartbeat failed & both active scenarios

The following table elaborates the scenarios of SystemController heartbeat failed and both database nodes isolated. This scenario is also called brain split.

|  |  |  |
| --- | --- | --- |
| Reasons | Scenario | Description |
| No business involved  SystemController failure & database HB failure | Both active and only one working | This scenario is working well, and it’s same as single point failure. The data which generated at the isolated point is non-business involved. The existing replication mechanism can be used. |
| Both active and switch | If all the message(s) from source non-context association, non-business association, then the existing replication mechanism can be used.  If the message(s) from source context association, the merge strategy should be under timestamp of the message. |
| Both active and working together | Both nodes processing its own message(s) and all the message(s) non-business association. The existing replication mechanism can be used. |
| Business involved  SystemController failure & database HB fauilure | Both active and only one working | This scenario is working well, and it’s same as single point failure. The data which generated at the isolated point is non-business involved. The existing replication mechanism can be used. |
| Both active and switch | This scenario is for both nodes failure in sequence manner, and message write to each side. Currently, there is no appropriate strategy to handle this scenario. Two alternate mechanisms can be used such as:  The existing replication mechanism might cause data inconsistence.  Merge the message(s) by timestamp order manually. |
| Both active and working together | Both nodes processing its own message(s) isolated and business involved for some message(s). Currently, there is no appropriate strategy to handle this scenario. Follow two alternate mechanisms can be used such as:  The existing replication mechanism might cause data inconsistence.  Merge the message(s) by timestamp order manually. |

Table: System Controller HB failure & network break (brain split)

1. RCV\_MSG\_CHECKER ---- table for message duplication check, it applied at each database server.

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Description |
| PKEY | INT | The primary key of the record. |
| LOCATION\_KEY | INT | The location key for the current connected active SynchEngine belongs to. The connected SynchEngine provides this info while it connects to server. |
| SRV\_ID | INT | The server id for the current connected active SynchEngine belongs to. The connected SynchEngine provides this info while it connects to server. The value usually should be: 1 --- indicate node 1; 2 -- indicate node 2 |
| MSG\_HID | INT | The high id of the received message. |
| MSG\_LID | INT | The low id of the received message. |

Table: rcv\_msg\_checker

1. STATION\_SRV\_TOKEN ---- table for recording currently active server(s), it applied only at Central Oracle database server for central SynchEngine(s) is being connected by station(s).

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Description |
| LOCATION\_KEY | INT | The location key for the current connected active SynchEngine belongs to. The connected SynchEngine provides this info while it connects to server. |
| SRV\_ID | INT | The server id for the current connected active SynchEngine belongs to. The connected SynchEngine provides this info while it connects to server. The value usually should be: 1 --- indicate node 1; 2 -- indicate node 2 |
| UPD\_TIME | DATATIME | The update time when refresh the status of current connected active SynchEngine. Currently, it will be refreshed at periodically rate 5 seconds. |

Table: station\_srv\_token

The following setctions details these two scenarios. In order to verify & validate the feasibility of the solution, We will use the following database name for example.

1. *TRASNACT* --- central database
2. *DT17* --- local database
3. DT17-CMS-01 --- local database server node 1
4. DT17-CMS-02 --- local database server node 2
5. *DT18* --- local database
6. DT18-CMS-01 --- local database server node 1

DT18-CMS-02 --- local database server node 2

##### Group proxy connection Strategy

Therefore, the message(s) came from only one source database server and the active SynchEngine will try to send message(s) to its group member based on the predefined configuration. The configuration info shown below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PKEY | NAME | HOSTNAME | ADDRESS | DB\_LINK\_NAME | LOCATION\_KEY | PRIORITY | STATUS |
| 1 | TRA\_DT18 | dt18-sms-01 | 192.168.0.181:5001 | TRA\_DT18\_AQ | 18 | 1 | 0 |
| 2 | TRA\_DT18 | dt18-sms-02 | 192.168.0.182:5001 | TRA\_DT18\_AQ | 18 | 2 | 0 |

The source active SynchEngine will connect to the high priority one of the destination SynchEngine based on the configuration settings above, however, if the selected SynchEngine switch to *PASSIVE* status, then try another one. As only one active at source side, no duplicate data will be sent.

DT17

DT18

Only SE1 will be selected based on the configuration settings.

SE1

SE2

SE1

SE2

SE1

0 1000

0 1001

0 1002

##### Message Enqueu Strategy

Duplicated message will be sent to central in this senario as the local database server(s) running well. All the data came from only one database source point and the message seqence IDs are duplicate also. With this happened, the duplicate checker table will be featured to verity and serve as a role to check duplicated message(s). Following uses central database *TRANSACT* and local database *DT17* to detail its procedures. At beginning the duplicate checker table has two initial records for location *DT17* show below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PKEY | LOCATION | SRV\_ID | Q\_NAME | MSG\_SID |
| 1 | TRA\_DT17 | 1 | 0 | 0 |
| 2 | TRA\_DT17 | 2 | 0 | 0 |

Then each message sent by DT17 will verity it MSG\_HID & MSG\_LID and will be applied to central database queue when and only when the message high & low IDs are greater than the current one which stored in the above table such as equation below:

*new.MSG\_SID >= current.MSG\_SID & new.Q\_NAME = current.Q\_NAME*

the procedure can be detail for instance as:

We set the active SynchEngine at server node 1,2 as: SE1, SE2.

1. *Assume that there were three messages in database and its message IDs are*

*MSG\_SID: 1000, 1001, 1002.*

1. *SE1 packed all these three messages and sent to central;*
2. *Assume one new MSG\_SID 1003 en-queue at this time;*
3. *SE2 packed all these four messages and also sent to central;*
4. *SE1’s messages will be applied into the queue of central as all its messages meet the above equation.*
5. *The first three messages of SE2 will be eliminated by the equation and the forth message will be applied successfully and current IDs updated*.

TRANSACT

DT17

0 1000

0 1001

0 1002

0 1000

0 1001

0 1002

0 1003

SE1

SE2

The red color items eliminated at server.

##### Download Control Strategy

1. **Both local SynchEngines Active with recovery from Central**

There are two data flow that messages will be sent from central to local one is that message(s) generated at central and need propagate to stations(s) and another is that message(s) generated by station(s) agent(s) when the station(s) database servers offline. All SynchEngine(s) running as active status at central database and being connected by station(s). Therefore, table *STATION\_SRV\_STATUS* is introduced to serve as status monitor for each connected SynchEngine from station. Location key and server id is provides by SynchEngine(s) while it connect from station to central and this info will be recorded in table: *STATION\_SRV\_STATUS.* Each connection will be refreshed periodically. The SynchEngine(s) running at central side is response to monitor/refresh its own active connection(s) status. The initial values for *DT17* while both SynchEngines connected and active at central shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PKEY | LOCATION | SRV\_ID | Q\_ NAME | UPD\_TIME |
| 1 | TRA\_DT17 | 1 | AUDIT\_DATA\_QUEUE | 2012-05-24 14:40:11 |

Based on the above table data, the selected SynchEngine is calculated as:

1. *If both status is well and update time within refresh interval, then the high priority one will be selected.*
2. *If one of the update time longer then the refresh interval, then regard it as bad and another one will be selected.*
3. *In normal scenario, only one record will be presented at the table for one location, and be selected.*

Therefore the token for sending message from central to local based on the above rules.

TRANSACT

DT17

SE1

SE2

### Data Reconstruction Module

This section depicts the detail architecture of the middle-ware sub-system which synchronize data in a two-way pattern in a distributed networking based database. The system consists of six modules to implement the exchange data through the network such as Gather Changes, Log Refine, Synchronization Dispatch/Trigger, Network Communication, and Data Update module. The diagram shown below illustrates the interaction between these modules.

### Threading and Concurrence Design Decision

### Data Consistency&Integrity Design Decision

In order to ensure data consistency and integrity while synchronizing data between heterogeneous database servers, the distributed two-phase transaction is being used to ensure the data consistency and integrity. The system used point-to-point communication in order to easy the interaction of the whole distributed system.

* All message(s) propagate in a FIFO sequence.
* ID for each Data package while traveral through network and uses it to confirm/validate from the source.
* A distributed transaction used for commit/rollback at both client&server sides.

### MMS Agent Message Design Decision

MMS message goes to message queue directly, MMS agent directly dequeue MMS message on CMS database.

## Related cots

The third-party libraries which the SynchEngine dependent are listed below:

* mcl -- for SynchEngine to interact with MySQL database at local MySQL station(s).
* ocilib -- as the existing ocl doesn’t has the AQ APIs, the SynchEngine need this library to interact with Oracle database at central station.
* zlib -- for compress/uncompress message while traveral through the network.

## Detail design

### Data Queue Module

For the detail design, please refer to MySQL\_Database\_AQ\_Library\_Design\_spec.docx

### Data Propagation Module

For the detail design, please refer to Distributed\_Database\_Propagation\_Design\_spec.docx

### Class diagram

This section details all the modules class view.

### Sequence diagram

This section presents the sequence diagram(s) for each module.

* Capture Change Module

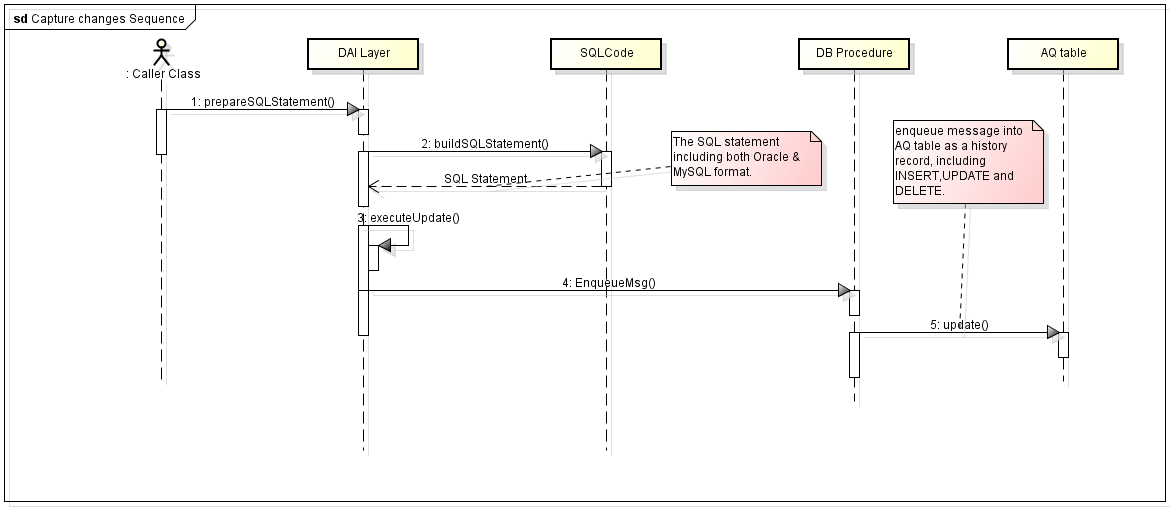


Figure: Capture Change Sequence

* Publish Data Module (deq-message)

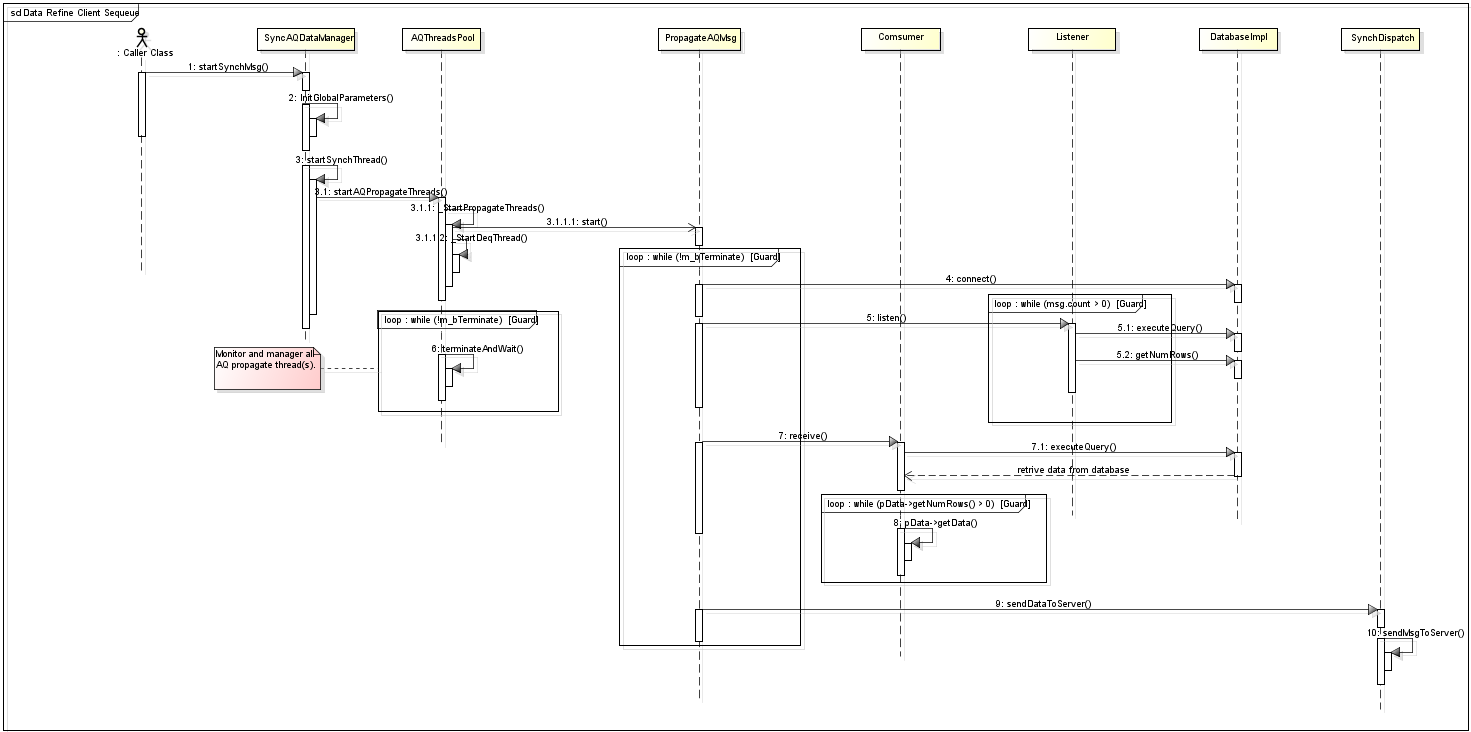


Figure: Log Refine (deq-message) Sequence

* Synchronization Dispatch Module

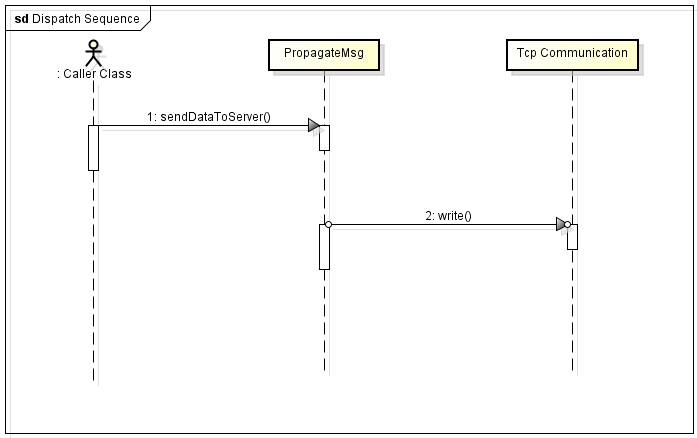


Figure: Synchronization Dispatch sequence

* Communication Module

For the communication module, there are two categories to interact with other station(s).

Send Message to Socket

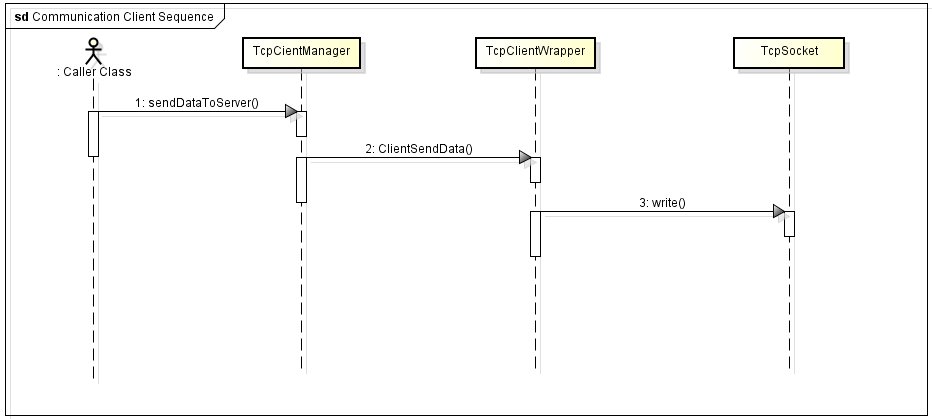


Figure: send message sequence

Read Message from Socket

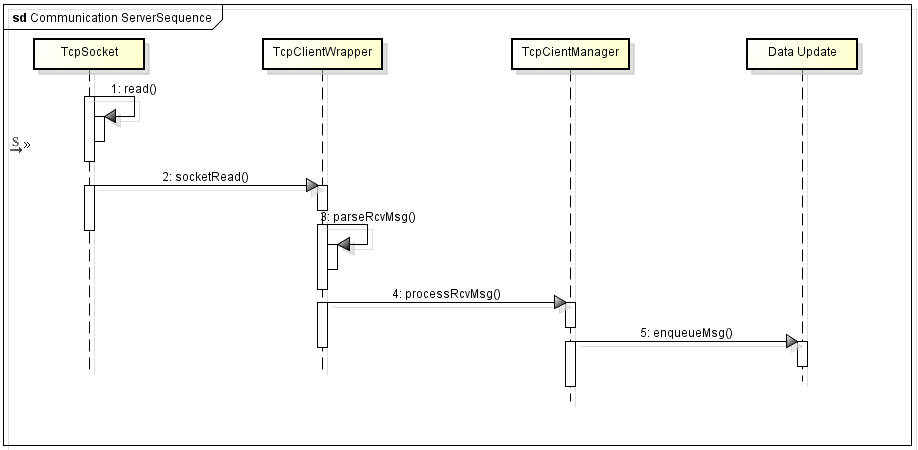


Figure: read message sequence

* Data Update Module

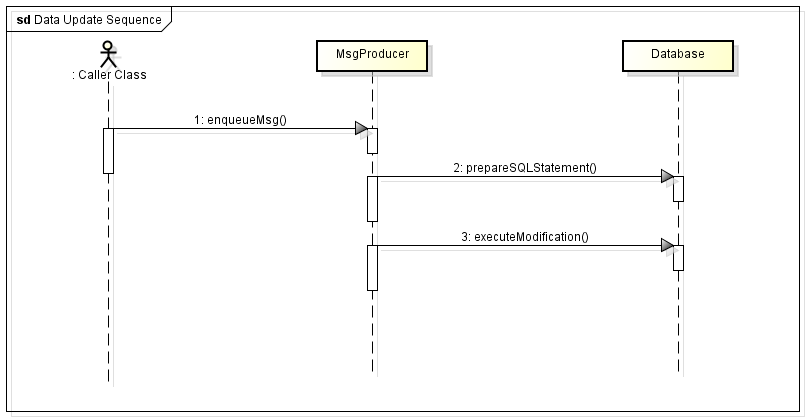


Figure: Data Update Sequeue

### State diagram(if necessary)

This section illustrates the overview state diagram of the SynchEngine system.

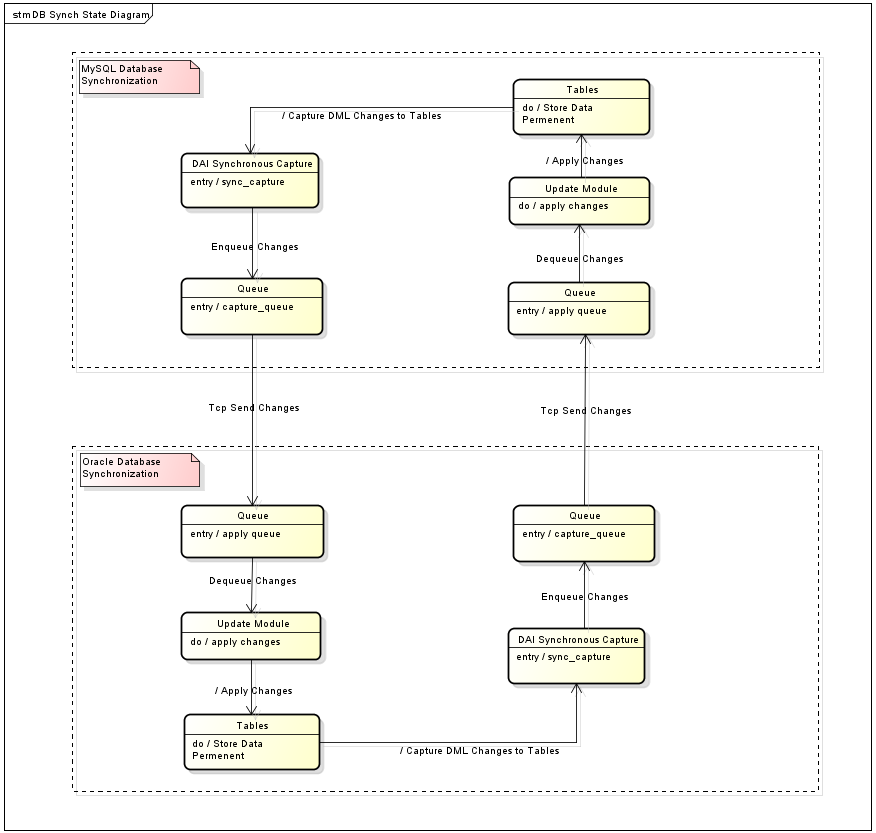


Figure: state diagram of the SynchEngine

### Component diagram

This section presents all the components which consist of the SynchEngine system to meet the requirements for data replication between heterogenoues database systems.

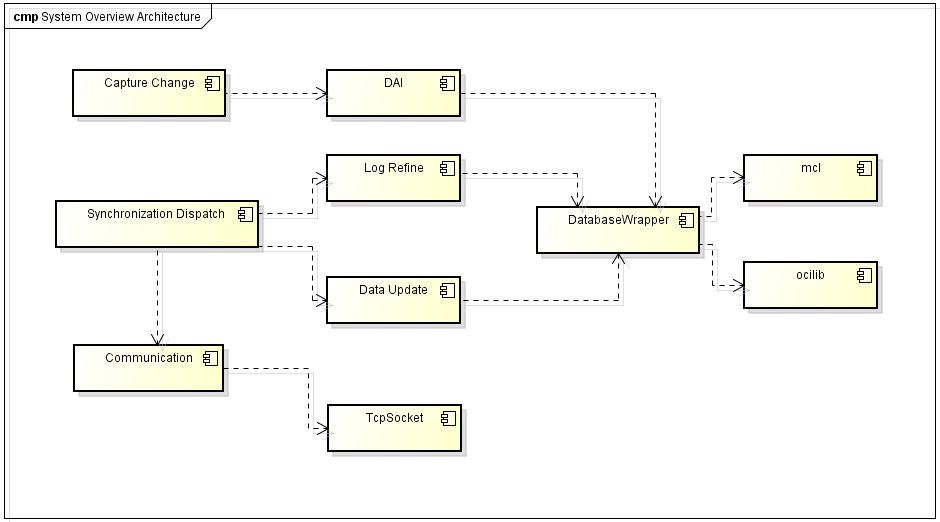


Figure: system componment diagram

The module Capture Change is dependent on some existing componment and database procedures, following diagram details the relationship of this componments.

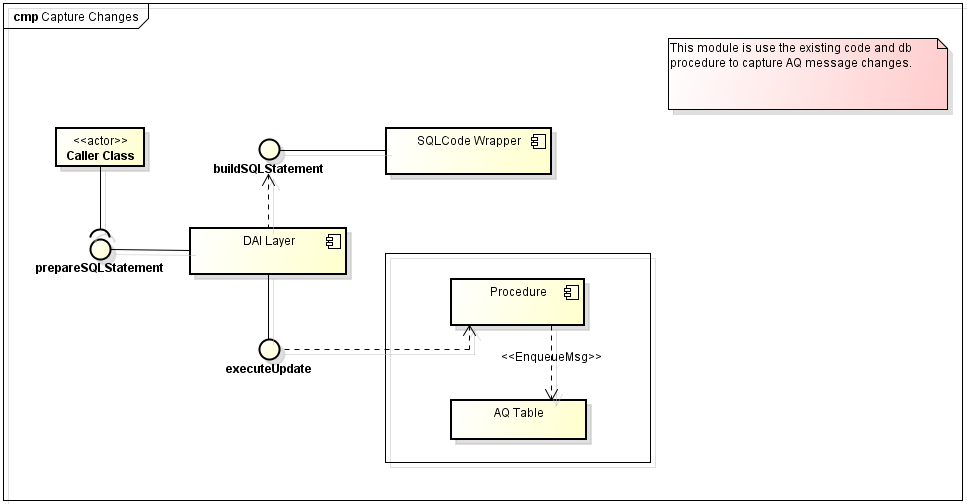


Figure: Capture Data componment diagram

### Degrade mode

### Design capacities

### Threading and concurrence

Based on the data replication flow, the design of threading & concurrence of the SynchEngine can be divided into two categoris.

* Multi-Thread

At local MySQL station(s), the type of data replication defined as below:

* INSERT, UPDATE, DELETE (1)
* GROUP (2)
* PUBLIC (3)

Both of (1), (3) message send to central DB, and the (2) message send to group member(s). In current design, the number of group member(s) is less or equal than 3, therefore in local MySQL station(s), multi-threading mechanism will be used to interact with each other. The number of thread(s) for one site based on the configuration *recipient* table. The number of group member(s) can be configurable by using *recipient* table.

* Multi-Process with Multi-Thread

At central Oracle DB, the SynchEngine deploy at each cluster node and each of the SynchEngine working separate to handle all the message request from station(s). The number of process based on the number of cluster node; and the number of thread per each process based on both the number of cluster node and station count.

The SynchEngine can serve as both Client & Server based on the configuration. For the configuration data, please refer to section 4.4.14. The following depicts the threading module and concurrence strategies which used in SynchEngine.

* As Client

|  |  |
| --- | --- |
| Thread Name | Thread’s task / Reason for separate thread |
| CProcessAQMsg | The purpose of this thread is to monitor the status of all the message propagate thread(s). It serves as a thread pool to startup all needed message propagate thread(s) based on the configuration recipient table for data synchronization. It’s also initialize the deq job thread. |
| CPropagateMsg | The purpose of this thread is to listen message from database and then propagate to destination station(s). |
| CProcessRcvMsg | The purpose of this thread is to process the message which receive from remote station(s) separately. |
| CTcpComSingleImp | The purpose of this thread is to monitor the tcp sockect for checking whether there is data need to read from or write to socket. |
| CDeqMsgJob | The purpose of this thread is to deq message from local queue table and apply to local DB periodically. |

* As Server

|  |  |
| --- | --- |
| Thread Name | Thread’s task / Reason for separate thread |
| CTcpConnectionListen | The purpose of this thread is to listen Tcp Socket whether there is client Tcp connection come in. |
| CSrvWorkMutilRcv | The purpose of this thread is to listen Tcp Socket whether there is data to read from the connected client. |
| CSrvWorkMutilSend | The purpose of this thread is to listen message from database and then propagate to the connected client. |
| CTcpConnectionPool | The purpose of this thread is to monitor & control all the thread(s) for all the connected client(s). It serves like a thread pool manager. |

### Error Handling

The following table lists the majority error handling which may occur in the system.

|  |  |
| --- | --- |
| Scenario | Handler |
| Enq-message to database failed because of database server down | Try three times and if still cannot success handle the operation, then connect to standby DB server. |
| Tcp send message failed because of network problem | Try to reconnect socket periodically until success connect to Tcp server and send the message again. |
| Memory leak for OCI library at Oracle side | Clean up and reinitilize Oracle client environment when it consumes 50% memory of the OS. |

### Failover

### Interface

### Message

This section defines the message format which traveral through network with each other. There are two kinds of messages in our synchronization system. For easy to parse and transfer of the message, the format of the message is defined as below:

MT QN OMI Priority Length

Message Header Message Data

Figure: message data structure

MT – type of the message.

Value:

0 – Message contains payload

1 – Message only contains message count

Length: 1BYTE

QN – queue name. Value: the target queue name. Length: 32BYTE

OMI – the source/original message ID. Value: unique ID. Length: 36BYTE

Priority: priority of the message. Value: 0–immediately; 1- fix time. Length: 1BYTE

Length: the length of the message. Value: unsigned integer size. Length: 4BYTE

The following tables show the details of the message format.

|  |  |  |
| --- | --- | --- |
| Size(BYTE) | Title | Description |
| 1 | MT | Type of the message. Value: 0x30 |
| 32 | QN | Queue name of the message belong to. |
| 36 | OMI | Message ID. |
| 1 | Priority | The priority of the message. |
| 4 | Length | The message length. |
|  | Data buffer | The message buffer |

Table 1: message format of payload

|  |  |  |
| --- | --- | --- |
| Size(BYTE) | Title | Description |
| 1 | MT | Type of the message. Value: 0x31 |
| 32 | QN | Queue name of the message belong to. |
| 36 | OMI | Message ID. (NULL) |
| 1 | Priority | The priority of the message. |
| 4 | Length | The message length. |
| 4 | Message count | The message count which processed. |

Table 2: message format of reply

|  |  |  |
| --- | --- | --- |
| Size(BYTE) | Title | Description |
| 1 | MT | Type of the message. Value: 0x32 |
| 32 | QN | Queue name of the message belong to. |
| 36 | OMI | Message ID. (NULL) |
| 1 | Priority | The priority of the message. |
| 4 | Length | The message length. |
|  | Data buffer | The message data buffer. |

Table 3: message format for create connection

In order to easy the interaction between the client(s) and server, a communication protocol is defined for data transformation. The protocol depicts as below:

PT DC DL

PL Data Header Data

STX Message Content ETX

Figure: structure of data communicate protocol

STX --- start of the text. Value: F8H. Length: 1BYTE

ETX --- end of the text. Value: FEH. Length: 1BYTE

PL --- length of the package. Value: the length of the package. Length: 4BYTE

PT --- type of the package. Value: 0 – request; 1 – reply. Length: 1BYTE

DC --- data compress option. Value: 0 – FALSE; 1 – TRUE. Length: 1BYTE

DL --- length of the data. Value: the length of the message data. Length: 4BYTE

From above definition, beside the message data, the total size of the message header is 12BYTE.

There are three types of packages which designed in our communication module, the package format and data description detailed as below:

|  |  |  |
| --- | --- | --- |
| Size(BYTE) | Title | Description |
| 1 | STX | Indicate the start of the message. (0xF8) |
| 4 | PL | Size of the package. |
| 16 | PID | The unique id of the package |
| 1 | PT | Type of the package. (0x00) |
| 1 | DC | Option for compress. (0x00 or 0x01) |
| 4 | DL | Size of the message data. |
|  | Data | The message data. (variable size) |
| 1 | ETX | The end of the message. (0xFE) |

Table 1: package format for sending

|  |  |  |
| --- | --- | --- |
| Size(BYTE) | Title | Description |
| 1 | STX | Indicate the start of the message. (0xF8) |
| 4 | PL | Size of the package. |
| 16 | PID | The unique id of the package |
| 1 | PT | Type of the package. (0x01) |
| 1 | DC | Option for compress. (0x01) |
| 4 | DL | Size of the message data. |
| 78 | Data | The message data. (only contains the message count which successfully processed) |
| 1 | ETX | The end of the message. (0xFE) |

Table 2: package format for replying

|  |  |  |
| --- | --- | --- |
| Size(BYTE) | Title | Description |
| 1 | STX | Indicate the start of the message. (0xF8) |
| 4 | PL | Size of the package. |
| 16 | PID | The unique id of the package |
| 1 | PT | Type of the package. (0x02) |
| 1 | DC | Option for compress. (0x00 or 0x01) |
| 4 | DL | Size of the message data. |
|  | Data | The message data. (variable size) |
| 1 | ETX | The end of the message. (0xFE) |

Table 2: package format for connection

### Dependencies

The SynchEngine system dependent on the following items:

* SystemController
* Oracle Database Server
* MySQL Database Server

### Access Control

### Configuration Information

The system based on two type of configuration data: config file & database table. Following details the configuration data.

* Configuration file

Serveral sections are defined in the config file, details as below:

[dbsyn\_queue]

QueueName – the queue name which defined for data synchronization.

CenterDBName – the central DB name which used at client side.

[dbsyn\_com]

MsgCompress – option for whether the message need compress or not while traveral through network.

PackageSize – the package size of the message while traveral through network.

[dbsyn\_srcDb]

SrcDbConStr – the source database connection string which need to monitor for message propagate.

SkipErrors – the error(s) which can be ignored when interact with local database.

[Log]

LogLevel – the log level which for log out to log file.

LogFileSize – the size for log file.

LogFileCount – the maxium log files.

LogFileName – the log file name.

* Database table

The system work with serveral configuration tables to define the needed configuration data. Details as below:

Table recipient

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Description |
| PKEY | INT | The primary key of the record. |
| NAME | VARCHAR | The name of the local database, also called agent/consumer name in application like *tra\_occ* etc. |
| ADDRESS | VARCHAR | The IP address for the local database, also the address of the running SynchEngine, the format is: 192.168.0.1:2005 |
| DB\_LINK\_NAME | VARCHAR | No use in the current implementation, just reserve it. |
| STATUS | NUMBER | The switch for the record enable/disable.  0 – enable; 1 – disable |
| TIMEOUT | NUMBER | No use in the current implementation, just reserve it. |
| PROP\_TYPE | VARCHAR | The type of the database. |

Table recipient\_group

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Description |
| PKEY | INT | The primary key of the record. |
| NAME | VARCHAR | The name of the local database, also called agent/consumer name in application like *tra\_occ* etc. |
| MEMBER | VARCHAR | The name of the group member for the column name. |
| DB\_LINK\_NAME | VARCHAR | No use in the current implementation, just reserve it. |
| STATUS | INT | The switch for the record enable/disable.  0 – enable; 1 – disable |
| TIMEOUT | NUMBER | No use in the current implementation, just reserve it. |
| PROP\_TYPE | VARCHAR | The type of the database. |

Table recipient\_info

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Description |
| PKEY | INT | The primary key of the record. |
| NAME | VARCHAR | The name of the local database, also called agent/consumer name in application like *tra\_occ* etc. |
| HOSTNAME | VARCHAR | The host name of the local machine. |
| ADDRESS | VARCHAR | The IP address & port of the local machine for SynchEngine. |
| DB\_LINK\_NAME | VARCHAR | No use in the current implementation, just reserve it. |
| LOCATION\_KEY | INT | The predefined location key for each station in the Transactive System. |
| PRIORITY | TINYINT | The predefined priority for the server node, the value current used the server id. |
| STATUS | INT | The switch for the record enable/disable.  0 – enable; 1 – disable |
| DB\_TYPE | VARCHAR | The type of the database. |

Table recipient\_channel\_cfg

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Description |
| PKEY | INT | The primary key of the record. |
| QUEUENAME | VARCHAR | The name of the queue which used to synch data between MySQL & Oracle, in application like *audit\_data\_queue* etc. |
| RECIPIENT\_NAME | VARCHAR | The recipient name of the remote location like *tra\_occ, transact* etc. |
| MSG\_COMPRESS\_OPTS | TINYINT | The message compress options while traveral through network.  0 – uncompress; 1 – compress |
| MSG\_PACKAGE\_SIZE | INT | The maxium size of package while traveral through network. |

### System Controller Group off-line Handling

### Database Design Changes/Extensions

In order to imitate and achive the same features of Oracle AQ mechanism. Some tables should be introduced to asistant the implementation. The table below lists all the tables used in the SynchEngine synchnization system.

|  |  |
| --- | --- |
| Table Name | Description |
| RECIPIENT | The purpose of the table is same as project C830. It’s for PUBLIC,INSERT/UPDATE/DELETE message. |
| RECIPIENT\_GROUP | The purpose of the table is same as project C830. It’s for GROUP message. |
| RECIPIENT\_INFO | The purpose of the table is to provides information of all recipients, such as recipient name, hostname, address etc. |
| AQ\_ENQ\_DUPLICATE\_CHECK | The purpose of the table is to serve as a duplicate checker while receiving message. |
| AQ\_DEQ\_TOKEN\_CHECK | The purpose of the table is to serve as the spin lock at the server side while multiple station’s SynchEngines interact with central simultaneously. It ensure that only one SynchEngine from station can download message from central. |
| AUDIT\_DATA\_QUEUE\_T | The queue table for audit data in TransActive System. |
| AUDIT\_DATA\_QUEUE\_T\_I | The dispatch table for audit data in TransActive System. |

Table: tables used in SynchEngine system

In order to realize the same results/behavior like the existing Oracle AQ, two tables was introduced at MySQL side. Details as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| Q\_NAME | VARCHAR | 30Bytes | Y | NULL | The name of the queue. |
| ENQ\_HID | INT | 4Bytes | N | 0 | The high bit of message id. |
| ENQ\_LID | INT | 4Bytes | N | 0 | The low bit of message id. |
| MSGID | VARCHAR | 36Bytes | N | UUID() | The unique ID of the message. |
| PRIORITY | TINYINT | 1Bytes | Y |  | The priority of the message. |
| STATE | TINYINT | 1Bytes | N | 0 | 1 – ready; 2 – processing;  3 – remove delay; 4 - done |
| SUB\_CNT | TINYINT | 1Bytes | N | 0 | 0 – local; 1 – central; 2 – others |
| EXPIRATION | DATETIME |  | Y |  | The message expiration time. |
| ENQ\_TIME | DATETIME |  | N | sysdate() | The enqueue time of the message. |
| ENQ\_UID | VARCHAR | 30Bytes | Y |  | The enqueue user of the message. |
| DEQ\_TIME | DATETIME |  | Y |  | The dequeue time of the message. |
| RETRY\_COUNT | INT | 4Bytes | Y |  | The retry count of the message. |
| SENDER\_MSGID | VARCHAR | 36Bytes | Y |  | The original ID of the message. |
| SENDER\_NAME | VARCHAR | 30Bytes | Y |  | The sender name of the message. |
| SENDER\_ADDR | VARCHAR | 20Bytes | Y |  | The address of the sender. |
| USER\_DATA\_1 | VARCHAR | 4000Bytes | Y |  | The message data. |
| USER\_DATA\_2 | VARCHAR | 4000Bytes | Y |  | The message data. |
| USER\_DATA\_3 | VARCHAR | 4000Bytes | Y |  | The message data. |

Table audit\_data\_queue\_t: queue table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| SUB\_ID | INT | 4Bytes | N |  | The id of the message subscriber. |
| SUB\_NAME | VARCHAR | 30Bytes | N |  | The name of the subscriber. |
| MSG\_SID | BIGINT | 8Bytes | N | 0 | The sequence id. |
| MSGID | VARCHAR | 36Bytes | N |  | The unique ID of the message. |
| STATE | TINYINT | 1Bytes | N |  | The unique ID of the message. |

Table audit\_data\_queue\_t\_i: en-queue table

## Safety function capability

## Testability

## Maintainability

## Impact Analysis

## Deployment considerations

There are several data synchronization technologies and methods in a distributed network, such as many-one, point-point communication. In our system, in order to reduce the centre DB performance load and problematic risk, it’s use the many-one strategy to synchronize data between the centre DB and the local DB. The following figure illustrates the system topological structure of the system.

Centre DB server

Local DB server

Local DB server

SynchEngine

SynchEngine

SynchEngine

Local DB server

SynchEngine

## Flow-on Effects

## Extensibility and Reuse

# gui-component design information

# Appendix A: Software module design review

| **DTL (C955) Design Module Review** | | | **Date:** | | |
| --- | --- | --- | --- | --- | --- |
| **Doc. No.:** | | | | | |
| **Subject :** | | | | | |
| **Reviewer** : | | | | | |
| **Status** : | | | | | |
|  |  |  | | |  |
| **Status :** | 1.Draft | 2. Conditional Accepted | | 3. Accepted | 4. Rejected |

| **No** | **Ref** | **Page No** | **Comments** | **Response** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Review Outcome (completed immediately following the review)

| **Signature of Reviewer**  **Name** |  | **Date:** |
| --- | --- | --- |
| **Signature of Independent Verifier**  **Name** |  | **Date:** |

Review Closeout (completed when review actions are completed)

| **All actions completed and closed** | **✓/🗶** | |
| --- | --- | --- |
| **Signature**  **Name** |  | **Date:** |