Database Synchronization with Symmetric DS

Primary mentor	Mike Seaton
Backup mentor	Justin Miranda
Assigned to	Nadeeshani Hewage

Abstract

Commonly, OpenMRS is deployed and implemented in resource-poor settings where internet connectivity is not 100% reliable. At the same time, many implementations in these environments want a unified system in which each facility has a common view of all data. To address this, OpenMRS has invested a lot of time and effort into the development of a Synchronization module that performs bi-directional synchronization across all configured servers. Partners In Health supports implementations in both Rwanda and Haiti that use this synchronization module to support local servers that provide fast, reliable access to data across multiple facilities.

The paradigm employed by the current synchronization module is one of API-level synchronization. All changes made via the API and persisted to the database via hibernate are captured, logged, and transmitted to the other servers in the network through a central "parent" node. This approach, although relatively successful, has a number of drawbacks. One is that it puts the burden of maintaining database synchronization on the application. This opens up a number of failure points and has proven fairly maintenance intensive. Another is that it does not allow for any direct manipulation of the underlying database, limiting certain system integration options, and complicating support and maintenance.

The goal of this project is to develop an alternative synchronization paradigm using SymmetricDS. SymmetricDS, from it's website, describes itself as "an asynchronous data replication software package that supports multiple subscribers and bi-directional synchronization. It uses web and database technologies to replicate tables between relational databases, in near real time if desired." Phase 1 of this project would be to build the necessary tools and configuration to successfully synchronize a single database table across 2 OpenMRS instances. Phase 2 would be to support all tables (via web configuration) in the OpenMRS schema.

Objectives

1. Set up two duplicate OpenMRS instances and configure SymmetricDS to successfully synchronize the "person" table.
2. Support bi-directional sync that supports periods of downtime, and which implements simple conflict resolution:
    - Likely involves customization of SymmetricDS via creation of a custom IDataLoaderFilter.
    - Likely involves matching of uuids rather than primary key ids if uuids are available on the table. Otherwise a last-update-wins approach.
3. Develop a new OpenMRS module
    - Ideally "embeds" symmetricDS as a library rather than depending on an external setup
    - Facilitates configuration by automatically setting up synchronization for all tables in the OpenMRS schema
    - Provides a UI for changing / enabling / disabling synchronization on individual tables.

Resources

SymmetricDS
Sync Module
Symmetric Thread 1
Symmetric Thread 2

Progress

A POC completed successfully by the end of GSoC 2012. Installing the developed module enables sync for your OpenMRS databases.

1. Setup two OpenMRS instances

- Done. Completed as specified on Wiki - Multiple Instances with the renamed .wars of omrsroot and omrsclient running on 2 ports for jetty. Independent two databases and 'omrsroot' and 'omrsclient' for sync, and having individual application data directories.

Steps to setup a root instance and a client instance for the project demo purpose

Folders referred to steps given below.

OpenMRS checkout location=OpenMRS-1.x.x
Default Application Data Directory=
Linux: /home/user_name/.OpenMRS

Windows7: C:\Users\your_name\AppData\Roaming\OpenMRS

Step 1: First rename your existing Application data directory to something else so that the setup will run all new. This applies if you have already been using OpenMRS before this trial.
Step 2: Make two copies of OpenMRS.war and rename them to root.war, client.war inside OpenMRs-1.x.x/webapp/target
Step 3: Go to OpenMRs-1.x.x/webapp,

```
mvn jetty:run -Dwebapp.name=root -Djetty.port=8042
```

Set up OpenMRS using the Advanced option where you can name your own database. Name the database to rootdb.
Step 4: Exit Jetty. Now you once again have a new Application data directory created. Rename this to something else. Say OpenMRS_root.
Go inside, copy all the folders and files. Create a new folder where you prefer. eg: OpenMRS-1.x.x/webapp/root. Paste the copied content inside this folder. So root is your new Application data directory for the root instance we are going to try.
Step 5: Open the runtime properties file inside this folder, paste the following line in it. We are letting the root-runtime properties know where to find its own application data directory, with its own modules seperated from another instance's ones.
application_data_directory=path.to.root.folder
Copy the edited runtime.properties file to OpenMRS-1.x.x/webapp from where you run the mvn jetty run command.

> *eg: application_data_directory=root*

Note: This path is relative to the new placement of root-runtime.properties. (Additionally, you may paste the same runtime.properties to OpenMRs-1.x.x /webapp/target where the .war actually resides, for safety :))
Cool, so the root is setup. Run the command

```
mvn jetty:run -Dwebapp.name=root -Djetty.port=8042
```

to see your working root OpenMRS instance!
Step 6: Repeat step3 with the client.war. eg:

```
mvn jetty:run -Dwebapp.name=root -Djetty.port=8047
```

And name your new database to clientdb
Step 7: Repeat step 4 with renaming to OpenMRS_client. Let the new folder you create be OpenMRS-1.x.x/webapp/client for example. So client is your new Application data directory for the root instance we are going to try.
Step 8: Repeat step 5 with application_data_directory=path.to.client.folder

> *eg: application_data_directory=client*

Now the client is all setup!
So by the end, you will have 2 OpenMRS instances you can run at the same time.As for my example,
root on localhost/8042, with rootdb, and appdata directory=root
client on localhost/8047, with clientdb, and appdata directory=client

2. Configure SymmetricDS for the two OpenMRS instances. Then Synchronize the Person table. (As with the 2.x release of Symmetric)

- Done. (minor issue reported)
- Configuring using properties files** omrsclient.properties** omrsroot.properties
- Inserting symmertic specific tables to root database using an .sql file. openmrs_sds_sql** Channels** Triggers** Routers** Trigger-router links

- Registering and starting nodes
- Send loads, push/(and) pull data.

3. IDatabaseWriterFilterAdapter extension point (As with the 3.x release of Symmetric)

Replacing IDataLoaderFolder from SDS 2.x, IDataBaseWriterFilerAdapter has been introduced in SDS 3.x. This is an adapter which we can use to intercept the data flow before writing to the target database. The algorithm proposed and discussed was to hold the to-be-synced-raw at the extension point, get its uuid, lookup the target database for a row with exact same uuid, if one such exists (raw has already been synced) eliminate from modifying the target row, if one such does not exist at target, make the PK id a null in the to-be-synced row so that once inserted to the target PK gets autoupdated (custom process for some tables). But due to the complexity of datamodel the algorithm was not evolved to match the full datamodel, along with its foreign key constraints. Developed extension was successful for tables such as person and a subset of OpenMRS tables.

4. Synchronizing all tables of OpenMRS database with Symmetric (As with the 3.x release of Symmetric)

An SQL script was written for assigning channels to OpenMRS tables. (detailed comments are on the SQL script)

- triggerroutersettings.sql
- Two node-groups - omroot (for root side nodes) and omchild (for client side nodes)
- Two node-group-links
    - data flow from omchild (source) to omroot (target) - child pushes
    - data flow from omroot (source) to omchild (target) - child pulls
- Single channel for all tables assigned.
- One trigger entry on sym_trigger table, for each table of OpenMRS. (This allows inserts, deletes and updates)
- Routers to define where data changes are taken to. 2 Routers to map root-to-client and client-to-root
- Trigger-router links to map each trigger to a router, upway and down ways. 101*2 trigger-router links

5. Embedding SymmetriDS in an OpenMRS module, resolving dependencies, troubleshooting and configuring Symmetric nodes to start gracefully with specific configurations matching the embedded approach. (As with the 3.x release of Symmetric)

Module : org.openmrs.module.syncsds (SymmetricDS Sync Module)

SymmetricDS being embedded to an application causes a new Jetty instance to be spawned on new Thread (and perform data carriage on new defined ThreadPools). SyncSDS module consists of a new webapp directory to contain a web.xml to start symmetric with a Jetty instance. Installing the .omod on OpenMRS will spawn the Jetty and start Symmetric node which maps to that OpenMRS instance and its database.

openmrs-root-*.properties carries an unassigned registration url, meaning that its sync url itself is the registration url, that any requesting nodes should register with, in order to get and send updates. Sync url is the url that any node should find ‘this’ node with, hence holds an identity to a node. Root is configured such that, on an initial setup of the trigger-router-settings.sql SQL script will be run on the root database, so that the specifically mentioned node groups, node identities, triggers, channels, routers, mapping links will be inserted on to the root database. On an initial load sent to the client, these settings will flow to the client as well. After the script runs, installs triggers and Symmetric runs, original 101 tables of OpenMRS increases to 132, due to the sym-* prepended tables inserted by Symmetric. If you find any requirement to skip this step of automation of SQL script been run, “auto.config. registration.svr.sql.script” property can be disabled on the root’s property file, followed by a manual execution of these sql statements found on the omod /src/main/webapp/triggerroutersettings.sql. Any configuration changes on node/node group names should result on the .sql file changes as well, commonly to both root & client. Auto registration allowing ability has been enabled for root, meaning that any client which requests registration from root will be allowed to register with. This can be disabled with the application of the real use case. Root also has been configured to send reloads to the clients as they register. (An initial load has also been configured to be considered as a reload.)

Client is configured such that if no node identity is found and it is not a root node (no registration url), to auto insert node, group, security and identity rows. Clients will not run any initial SQL scripts, but manage with the above. Registration number of attempts has been left at infinity, leaving the client to keep polling for a root node to register with.

Please find all configuration files also on the module svn location.

6. Known issues

- Any changes to configurations on property files either on root or on client (ipaddresses, ports, database urls, database passwords) require the omod to be rebuilt.
- Since the location for webapp/resourses to be held for an uploaded omod differs on whether OpenMRS.war is deployed on Tomcat or Jetty, user requires to notify on global properties which category he falls to.
- Module activator gets called pointlessly after the module is stopped and results in an exception harmless to neither OpenMRS nor Symmetric.

7. Points to note before installing the .omod

- Configure personal settings such as ip, port for Symmetric to run, your database specific property settings on the root and client property files.
- Changes to node identity names, nodegroup names should be reflected on the SQL script on omod/src/main/webapp as well.
- Prior to installing your omod, include the following global properties to your database.
    - ***syncsds.symmetric.nodeGroup*** *for root only : ‘root’*
    - ***syncsds.symmetric.nodePort*** *for root : ‘8087’, for client : ‘8835’ (changes to values should result on property file configurations too)*
    - ***syncsds.jettyUser*** *for OpenMRS jetty users: ‘true’, for Tomcat users: ‘false’*
- Deploy the OpenMRS war giving the following JVM option
    - *-Djmx.http.console.for.embedded.webserver.enabled=false*
    - eg: (for Jetty users)

```
mvn jetty:run -Djetty.port=8042 -Djmx.http.console.for.embedded.webserver.enabled=false
```

8. Youtube Video of synchronizing few patients

You can find a brief demonstration on [0] and a much more descriptive version on [1]

[0] http://www.youtube.com/watch?v=5b2BSqHUImQ&feature=youtube_gdata
[1] http://www.youtube.com/watch?v=vTdkSeoW7yo&feature=plcp

Timeline

The testing and documenting for early stages will go inline with development.

Time Period	Suggested Workflow	Progress
23rd April - 21st May	Community Bonding Period	Completed
21st May - 31st May	Bi-directionally synchronize person table on 2 OpenMRS instances.	Completed
1st June - 10th June	Develop OpenMRS module to embed SymmetricDS jar.	Completed
11th June - 9th July	Develop means to identify & resolve conflicts arising through bi- directional synchronization. Implement IDataLoaderFilter. The module will include developed filter functionality at the end.	Completed for a subset of tables Issues due to complex data model discussed
9th July - 16th July	Configure SymmetricDS for synchronization of all tables in OpenMRS schema. Include this in the on-going module.	Completed
16th July - 5th August	Develop the UIs for authenticating & configuring SymmetricDS. Develop the UI to give the user ability to enable/disable and thus control synchronization for individual tables.	Allocated for troubleshooting module encapsulation and extension development - mentioned additional
15th August - 13th August	Test and document module	Completed
13th August - 20th August	Getting feedback from community. Test on multiple physical nodes for network latency and other practical issues. Implement security and monitoring functionality (If time permits as suggested above)	Tested on 2 physical nodes with 2 operating systems
Additional	Client & root specific configuration to get OpenMRS databases setup for Symmetric running on a new Jetty instance. Includes a web.xml, sql script and configuration on properties files to fit the embedded approach, differing from the standalone approach	Completed

UI Mockups Proposed

Linking page from Personal Space