

Distributed Polling System Message Routing and Transformation

Version 1.0

Table of Contents

1. INTRODUCTION.....	3
1.1. PURPOSE OF THIS DOCUMENT	3
1.2. SCOPE OF THE DOCUMENT	3
1.3. TARGET AUDIENCE	3
2. DPS UNDERLYING MESSAGE ROUTING AND TRANSFORMATION ARCHITECTURE.....	3
3. EVENT TABLE AND TRIGGER	6
4. MESSAGE PARSING	7
5. VOTING SYNTAX AND PARSING MECHANISM	8

1. Introduction

This document is an outcome of DPS project implementation phase to delineate the underlying mechanism of message transformation and routing. Applications involved in DPS were fulfilling their functionalities through the means of messaging communication provided by Jboss middleware. Applications are unaware of the technologies used for messaging, rather acts as a provider and consumer of messages, served by middleware.

1.1 Purpose of this Document

This document describes methods and techniques used for message transformation and routing along with a little bit of example code that has constructed DPS. Even this document delineates message flows and the way of working in underlying technology arena according to DPS architecture.

1.2 Scope of the document

This document describes underlying message flows, transformation and routing mechanisms implemented in DPS.

1.3 Target Audience

Below is the list of intended audience for this document:

- DPS project team
- Customers and Supervisors
- SCORE reviewers

2. DPS underlying message routing and transformation architecture

Figure 2-1, depicts that DPS consists of Web-DPS, Email Server, and SMS Gateway as applications to fulfill DPS functionality and Jboss middleware as the messaging provider. This also portrays the underlying means of communication and message transfer/flow among these applications.

Using Web-DPS application GUI (visible to the end-user), a poll creator can create a poll which is then captured in Web-DPS database storage and an event is constructed in '**Middleware_Event_Table**'. A JDBC listener then polls this table and triggers a subscribed/consumer service '**receiveParseEvent**' implemented at Jboss middleware hub.

This service then parses the message and provides the details to '**businessProcessController**' service which then control the business process (interaction with SMS Gateway or Email server and so on) based upon business logic by using custom built granular services at middleware hub (plaintextEmailSender, attachmentEmailSender, SMSSender, updatewebdpsArchiveTable, PDFGenerator).

Similarly, for receiving votes from SMS Gateway, we use JDBC adapter listener and through the means of **receiveparseResponseSMS**, **updateVoteDPS** middleware services and **captureVoting()** stored procedure API (please refer DPS_Algorithm.doc for details), we update SMS votes at Web-DPS for vote calculation.

For Email votes, using Jboss product suite provided Email adapter listener, we poll response@dps.mdh.se email account where all voters respond with their votes. Using **receiveparseResponseEmail**, **updateVoteDPS** middleware services and **captureVoting()** stored procedure API (please refer DPS_Algorithm.doc for details), we update the vote at Web-DPS application for vote calculation and announcement.

Design time consideration:

During designing custom polling table (**Middleware_Event_Table**, **SMS_Vote_Event_Tbl**), we had two options:

- Construct a column that will contain the triggered message in XML format **or**
- Specify each message parameter with a corresponding column in the table.

Considering limited bounded time-frame, we opted for second approach, to use a column for each message parameter. Because we were requiring construction /usability of two services

1. Converter from data stored in database table to XML message
2. XML parser at middleware hub

Note: Algorithms for vote calculation is described in details in a separate document **DPS_Algorithm.doc**. Procedure/steps for Vote announcement is typically same as intimating poll details to members/users.

Vote announcement also inserts events in a polling table **'Middleware_Poll_Announce'**. By using JDBC listener, businessProcessController, plaintextEmailSender, attachmentEmailSender and SMSSender middleware services, we intimate voting results to the members/users through their preferred means of communication (SMS and Email).

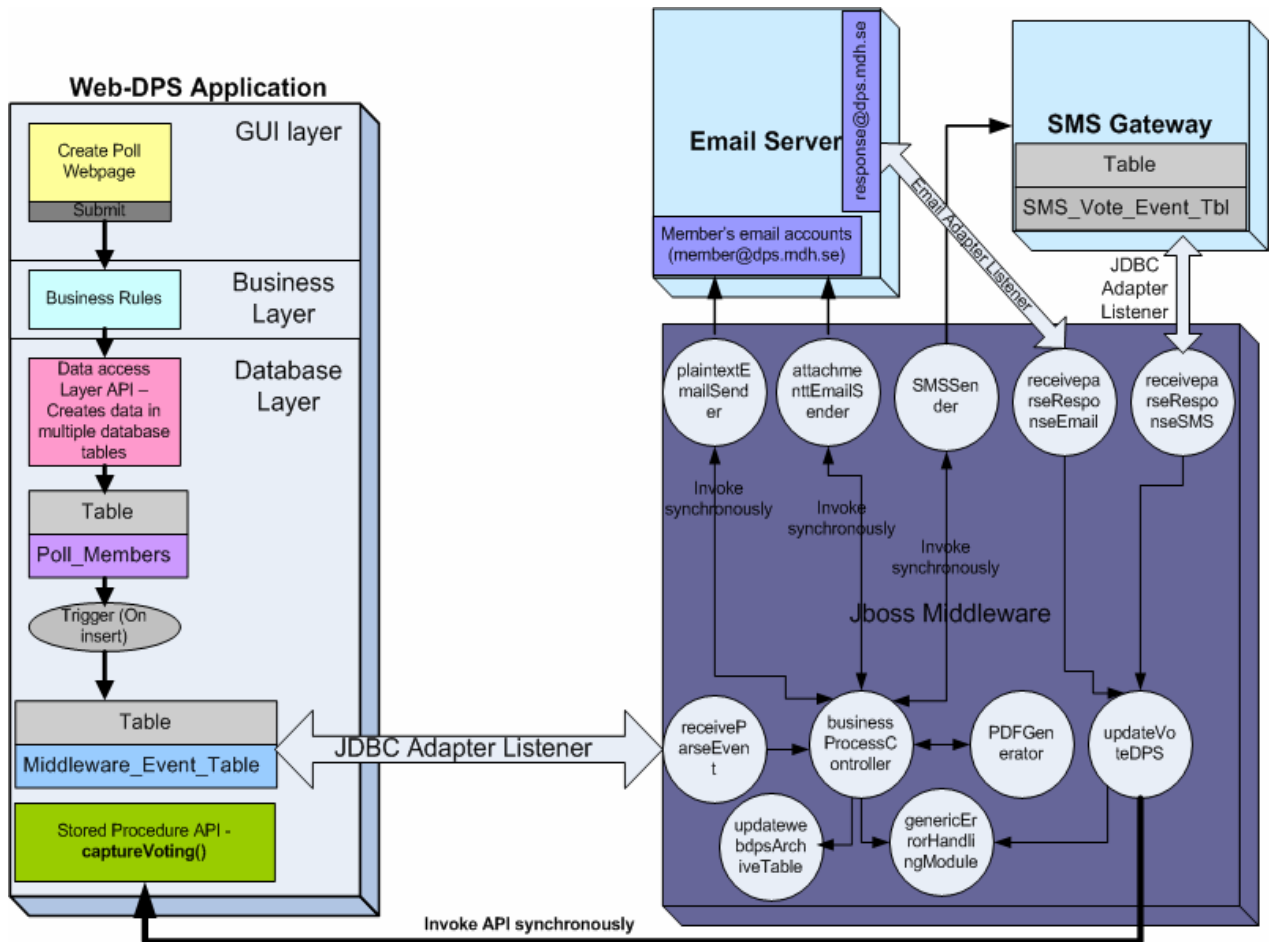
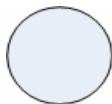


Figure 2-1: Underlying architecture of message routing and transformation



-- Reusable services at Jboss middleware layer where each service fulfills a specific functionality (also can be called as **components** in component based software architecture) that implement service oriented architecture (SOA) and loose coupling.

Example:

attachmentEmailSender middleware service sends business information details to email recipient as Password protected PDF attachment;

plaintextEmailSender sends business information as plaintext email content to email recipient;

PDFGenerator generates password protected PDF file with the business information details

SMSSender invokes SMS Gateway to send SMS via SMSC

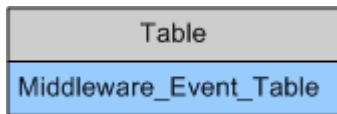
BusinessProcessController performs business process orchestration/choreography
genericErrorHandlerHandlingModule implements generic reusable error handling service

receiveParseEvent receives, parse and validates the new poll message

receiveParseResponseEmail receives, parse and validates the email message

receiveParseResponseSMS receives, parse and validates the SMS message

updateVoteDPS invokes captureVoting stored procedure API to update vote in Web-DPS
updatewebdpsArchiveTable updates the archive table **Middleware_Archive_Table** placed at Web-DPS application for event archiving



-- 'Middleware_Event_Table' and 'SMS_Vote_Event_Tbl' are the custom tables constructed as per messaging needs (i.e those all columns have been incorporated that is required by any target application). All other tables (poll, poll_answer, poll_members, poll_option, poll_response, poll_results, employee, user_corporate_account, user_availability, user_password) are the system tables used for Web-DPS application functionality.



-- Adapter listener ships with Jboss product suite as COTS (commercial of the shelf) package. This listener polls the targets in a constant interval (configurable), pulls out events from the targets and delivers to the subscribed/consumer service at middleware hub.

Description:

Step-wise details for New Poll intimation:

1. As depicted in Figure 2-1, once a poll creator creates a poll by accessing the **createPoll** webpage of Web-DPS application and clicks on **submit** button, after business validation and using data access layer API, the poll details are inserted in (poll, poll_option and poll_members tables).
2. During creation of the poll details in the database, we insert in poll_members table at last. Because once a row is inserted in poll_members table, a trigger '**POPULATE_MIDDLEWARE_EVENT_TABLE**' is executed which is attached with the poll_members table (otherwise if the trigger is executed beforehand, we wont get all required details from related other tables(poll, poll_option) which are yet to be committed with data).
3. The trigger inserts a row in the 'Middleware_Event_Table' for each member involved in the poll. That means there is one event for each member to whom notification will be intimated through their preferred means of communication (SMS or email).
4. Jboss JDBC adapter listener picks up the event from 'Middleware_Event_Table' and delivers it to the subscribed/consumer service **receiveparseEvent** at middleware hub. This service parses and validates the message with all required parameters for applying business logic on them and delivers to **businessProcessController** middleware service.
5. **businessProcessController** service performs the process choreography or so called process orchestration based upon business rules. It finds out the preferred means of communication medium (SMS or Email) with the member/user and invokes appropriate service (**attachmentEmailSender**, **plaintextEmailSender**, **SMSSender**) to intimate members. It also archives a processed event in Middleware_Archive_Table in Web-DPS database using **updatewebdpsArchiveTable** middleware service.
6. **businessProcessController** service uses **PDFGenerator** service to generate password protected PDF files with the business information details.

Step-wise details for Email Vote capturing:

1. All members/users responds by email to a specific email recipient response@dps.mdh.se
2. An email adapter listener polls this email account and once it receives any new email, it delivers to subscribed service **receiveParseResponseEmail** that parses the message and delivers to **updateVoteDPS** middleware service.
3. **updateVoteDPS** service invokes **captureVoting** stored procedure API synchronously to update voting response at Web-DPS application.

Step-wise details for SMS Vote capturing:

1. All members/users votes by SMS to a specific MSISDN (Mobile Subscriber Integrated Services Digital Network Number) acting as SMSC (for prototype purposes).

2. This voting information is stored in a custom database table (SMS_Vote_Event_Tbl) placed in SMS Gateway
3. Using Jboss JDBC adapter listener, we poll this event from **SMS_Vote_Event_Tbl** and the adapter delivers this message to **receiveParseResponseSMS** middleware service.
4. **receiveParseResponseSMS** service parses the message and delivers to **updateVoteDPS** service which invokes **captureVoting** stored procedure API synchronously to update voting response at Web-DPS application.

3. Event table and trigger

As specified in Figure 2-1, custom built `Middleware_Event_Table` and the trigger used in Web-DPS to generate new poll events are as follows:

Middleware Event Table Structure Explanation

```
CREATE TABLE `Middleware_Event_Table` (
  `row_id` int(11) NOT NULL auto_increment,
  `originator_email` varchar(40) NOT NULL,
  `topic_id` varchar(10) NOT NULL,
  `SMS_msg` mediumtext,
  `EMAIL_msg` longtext,
  `status` char(1) default 'P',
  `isPDF` varchar(5) default 'YES',
  `isAnonymous` varchar(5) default 'NO',
  `No_of_voting_options` varchar(5) default NULL,
  `voting_options` longtext,
  `subscriber` longtext,
  `MSISDN` varchar(15) default NULL,
  `priority` varchar(10) default NULL,
  `event_date` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `topic_name` varchar(20) default NULL,
  `poll_type` varchar(20) default NULL,
  `availability` varchar(10) default 'EMAIL',
  PRIMARY KEY (`row_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;
```

Row_id: unique id generated each time a new poll is inserted and used as a key value

originator_email: poll creator email id

topic_id: it's the poll_id unique for each poll and this id is sent to the members/users during poll intimation

SMS_msg: it's the SMS string used for sending SMS (First 160 characters will be taken)

EMAIL_msg: Business message, will be taken while sending email message (either as text or as PDF attachment)

Status: hardcoded 'P'. Used by JDBC listener to identify its ready for poll

isPDF: this parameter will suggest whether email should be sent as PDF attachment or as normal text message

isAnonymous: Whether member/user can vote anonymously or not

No_of_voting_options: it's used for different type of voting mechanisms (yes/no, one choice, multiple choice and priority) and it says how many options to be generated during PDF generation or to be specified during normal email text sending

Voting_options: this parameter contains different options as a normal string with '##' separator. So when the PDF is generated, tokenizing is based on '##' to differentiate options

Subscriber: recipient of email message

MSISDN: Mobile Number of recipient

Priority: Its message severity—High, medium or low

High: Intimate via both SMS and email

Medium: Intimate via SMS

Low: Intimate via Email

event_date: timestamp on the time this event is inserted

topic_name: A unique string by which members/users can vote.

Members are given two unique value topic_id and topic name. He can type any of these during voting, so that system can understand against which poll the vote has to accept (if multiple open votes are there)

poll_type: Values are: yes/no, one choice, multiple choice and priority

availability: Preferred means of communication to the members

Trigger

```
CREATE DEFINER=`middleware`@`%` TRIGGER
`jbpmbd`.`POPULATE_MIDDLEWARE_EVENT_TABLE` AFTER INSERT ON
jbpmbd.poll_members FOR EACH ROW
begin
insert into MIDDLEWARE_EVENT_TABLE
(originator_email,topic_id,topic_name,poll_type,SMS_msg,EMAIL_msg,isPDF,isAnonymous,No_of_voting_options,voting_options,subscriber,MSISDN,priority,availability)
(select b.corporate_email,a.poll_id,a.poll_name,a.poll_type,a.poll_details_sms, a.poll_details_email,
a.is_attachment, a.poll_anonymous,
(select count(*) from poll_option where poll_id=new.poll_id),
(select group_concat(distinct poll_option order by poll_option DESC SEPARATOR '##') from
poll_option where poll_id=new.poll_id),
(select y.corporate_email from user_corporate_account y where y.user_id=new.user_id),
(select corporate_phone_num from user_corporate_account where user_id=new.user_id),
(select poll_severity from poll where poll_id=new.poll_id)
(select availability from user_availability where user_id=new.user_id and status='ACTIVE')
from poll a,user_corporate_account b where a.user_id=b.user_id and a.poll_id=new.poll_id);
end;
```

4. Message Parsing

Receiving and Parsing of message at *receiveparseEvent* service

```
Map<String,Object> rowData =(Map)message.getBody().get();
StringBuffer results = new StringBuffer();
for (Map.Entry<String,Object> curr : rowData.entrySet()) {
System.out.println("ENTERED IN PARSING MESSAGE FROM SOURCE");
if(curr.getValue()!=null)
{
if(curr.getKey().toString().equalsIgnoreCase("originator_email"))
originator_email=(String)curr.getValue();
else if(curr.getKey().toString().equalsIgnoreCase("topic_id"))
topic_id=(String)curr.getValue();
else if(curr.getKey().toString().equalsIgnoreCase("SMS_msg"))
sms_msg=curr.getValue().toString();
else if(curr.getKey().toString().equalsIgnoreCase("EMAIL_msg"))
{email_msg=curr.getValue().toString();pdf_email_content=email_msg;}
else if(curr.getKey().toString().equalsIgnoreCase("isPDF"))
{is_pdf=curr.getValue().toString();System.out.println("****ISPDF****"+is_pdf);}
else if(curr.getKey().toString().equalsIgnoreCase("isAnonymous"))
is_anonymous=curr.getValue().toString();
else if(curr.getKey().toString().equalsIgnoreCase("No_of_voting_options"))
no_voting_options=curr.getValue().toString();
else if(curr.getKey().toString().equalsIgnoreCase("voting_options"))
voting_options=curr.getValue().toString();
else if(curr.getKey().toString().equalsIgnoreCase("subscriber"))
recipient=curr.getValue().toString();
else if(curr.getKey().toString().equalsIgnoreCase("MSISDN"))
phone_num=curr.getValue().toString();
else if(curr.getKey().toString().equalsIgnoreCase("priority"))
```

```

        message_priority=curr.getValue().toString();
    else if(curr.getKey().toString().equalsIgnoreCase("availability"))
        user_availability=curr.getValue().toString();
    else if(curr.getKey().toString().equalsIgnoreCase("poll_type"))
        { poll_type=curr.getValue().toString();
          if(poll_type.equalsIgnoreCase("1") || poll_type.equalsIgnoreCase("2"))
poll_type="yesno";
          else if(poll_type.equalsIgnoreCase("3")) poll_type="multiple_choice";
          else if(poll_type.equalsIgnoreCase("4")) poll_type="priority_choice";}
    }
}

```

5. Voting Syntax and Parsing Mechanism

Syntax of voting through EMAIL and SMS

Members can vote by using the unique **topic name** string or the **topic id** that ships with the business message in SMS or email message to the member.

For anonymous voting, members can specify either 'A' or 'ANONYMOUS'.

So, for example, if a member wants to vote for **YES** for **YES/NO** voting for a poll, where **topic id** is 183 or **topic name** string is 'memberSelection',

Voting string is :

(For anonymous voting)

Using topic id

- 183##YES##ANONYMOUS or 183##NO##ANONYMOUS
- 183##YES##A or 183##NO##A
- 183##yes##a or 183##no##a
- 183##yes##anonymous or 183##no##anonymous

Using topic string

- memberSelection#YES##ANONYMOUS or memberSelection ##NO##ANONYMOUS
- memberSelection##YES##A or memberSelection ##NO##A
- memberSelection##yes##a or memberSelection ##no##a
- memberSelection##yes##anonymous or memberSelection ##no##anonymous

(For un-anonymous voting)

No need to specify the third part

Using topic id

- 183##yes
- 183##YES
- 183##no
- 183##NO

Using topic string

- memberSelection##yes
- memberSelection##YES
- memberSelection##no
- memberSelection##NO

For multiple choice or priority voting, the syntax will as below:

If options are 1,2,3 and 4 for poll id 183:

Voting syntax: 183##1,3,2

For Multiple choice: 1,3 and 2 have equal precedence

For priority: 1 is highest priority, then 3 and 2 is the lowest priority

But the syntax is same for both these type of voting.

Voting Through Email Mechanism

Members need to vote with the above depicted syntax in two different ways as below:

- A member can directly compose an email with the above portrayed **syntax specifying vote in subject line** only. It does not matter whether there is any email body message or not. System reads only the email subject.
Note: Definitely email recipient should be correct (response@dps.mdh.se)
- If the poll intimation is through PDF attachment, the voting options are specified in the PDF. Member needs to select among the options in the PDF and no need to compose any email or do extra writings.
Then a click on the **send** button in the PDF will do the voting.
When submitted, system will take care of reading details from PDF submission and will send an email to response@dps.mdh.se with the subject line in above depicted syntax.

Email adapter listener reads the email message from response@dps.mdh.se and segregates each portion of the message (subject, from, to, body and so on).

To tokenize and to update at Web-DPS, we have used the below code fragment:

```
final String DELIM2="##";
String topic_id=null;
String option=null;
String anonymity=null;
String response_values[] = new String[3];
st = new StringTokenizer(emailsubject, DELIM2, false);
    while (st.hasMoreTokens())
    {
        response_values[i]=st.nextToken();
        if(i==0) topic_id=response_values[i];
        else
if(i==1){option=response_values[i];option=option.toUpperCase();}
        else if(i==2) anonymity=response_values[i];
        i++;
    }

    if(anonymity!=null)
    {
        if(anonymity.equalsIgnoreCase("A")
anonymity.equalsIgnoreCase("ANONYMOUS"))
        anonymity="YES";
        else anonymity="NO";
    }
    else anonymity="NO";
```