**System architecture and requirements**

**The aim of the project is to create the prototype of system for a distributed data management within existing RS Doc application.**

The main technical requirement states to use open source products for enterprise development. In spite of that the products were chosen among others are Spring Core 3.1.3, Spring Integration 2.2.1 modules, ActiveMQ JMS provider 5.8.0 and Postgres 9.2 rdbms.

The functional requirements are:

- Possibility to transfer files between distributed RS doc instances

- divide files by chunks during file transfer process to adjust provider channel speed and quality

- Possibility to call from the third part code due to the provided API, RS doc in our case

- Automatic connection reconnects on channel provider failure with message redelivery

- Possibility to synchronize several slave distributed RS doc instances simultaneously according to the master state

- Possibility for application modules to be loosely coupled, reliable, and asynchronous

- Logging possibilities during message delivery from master to slave’s hosts, with message acknowledge

- Possibility to secure message delivery

According to the requirements the system was built on the top of Spring integration module and utilizes JMS technology as a transport layer between distributed RS Doc instances.

The project utilizes ActiveMQ JMS provider with Postgress database as a pluggable message store. Spring integration [SI] module is used as a free light weight replacement instead of the commercial ESB provider tool.

SI centralizes messaging application business logic on distributed RS DOC instances and provides convenient means to communicate with different kind of data sources, as well as gateway components to communicate with third party applications.

Architecturally the application prototype consists of the following sub modules; according to the set functions they're provided:

a) Sender, responsible for message delivery, splitting in the case of need, routing and logging of out coming messages

b) Receiver, responsible for the incoming message delivery, aggregation, construction of notification message.

c) Monitor, responsible for the calculation of successful and unsuccessful messages delivery between different slave and master nodes, logging undelivered messages to the destinations.

d) JMS message provider

The modules and tasks to be planned and added in the nearest future:

a) The module for a notification on business error arisen under RS doc message processing

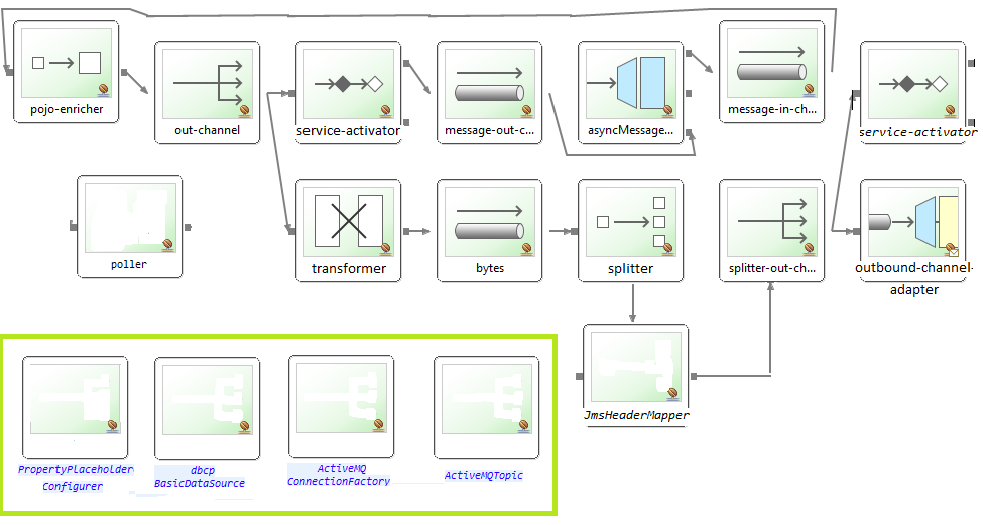
b) The module on custom message resend on undelivered messages

c) The module for monitoring the state of JMS provider, in the case of topic/queue persistent storage limit, etc. with email message notification for tech support

d) Redesign the existing architecture of single JMS provider for all RS DOC instances with standalone for every distributed RS doc instance

e) The module for the existing Web Services module integration in within system, to provide unique interface to the clients

***The Sender component the workfllow definition and explanation***

******

The definition of components used:

*- message-in-channel, bytes are channel components to*decouple the sender and receiver components

- out-channel, splitter-out-channel is a publish-subscribe-channel *to*decouple the sender and receiver components

*- asyncMessageGateway is a gateway component that define API interface to expose ESB [Spring Integration] functionality to the external system [RS Doc]*

- *pojo-enricher*is a channel component to enrich the incoming message with an additional information

- messageTransformer is a transformer component, to reorganize message structure

- MessageSplitter is a flow component that provides splitter functionality

- outbound-channel-adapter is component to publish to JMS topic/queue

- service-activator is an endpoint component that invokes a method on a bean when a message arrives at an input channel

-  correlationIdJmsHeaderMapper is a custom mapper to map attributes between internal ESB structure and external JMS provider

- poller is an utility component used to schedule pull data process from inbound components, currently doesn’t used probably as the only inbound component we had previously was inbound-file-adapter component.

The order of processing the income call from third party code is next:

1. *‘asyncMessageGateway’ component receives a call from third party code [RS Doc], the component provides API in form of the Java interfaces and transforms parameters in API methods to the internal SI message. On instantiation the component registers itself with two channels for inbound and outbound messages. Outbound message channel serves as confirmation of message processing by SI framework. Currently API provides one method that takes two parameters: String that presents message name in external system and byte array that respectively is a content of the message in external system.*
2. *‘pojo-enricher’ component is used as a means to add new parameter to SI header that denotes message name from Parcel domain object*
3. ‘out channel’ is a SI publish-subscribe channel that delivers internal message to two recipients simultaneously. Here we got message separation in two ways; the first one is used as a feedback confirmation for gateway component and logging facilities and the second as a preparation to the message to delivery via JMS outbound adapter.
4. ‘service-activator component’ [out-channel] is used as test case for figure out SI internal message structure during prototyping, could be removed afterwards in the case of need
5. ‘message-out-channel’ serves as a confirmation end point on message processing to the gateway component.
6. ‘transformer’ components extracts byte array from the incoming SI messages and creates new message with the byte array as a payload and just the same headers.
7. ‘splitter’ component splits incoming message depending on the message and the chunk size configuration parameter. Additionally it adds three header parameters [Sequence number, correlation ID, Sequence size], to denote that message was divided for the aggregator component on the client side.
8. ‘jms-header-mapper’ map message properties from internal SI message structure to the JMS structure, ‘correlation id’ in our case.
9. ‘outbound-channel-adapter’ publishes message to the JMS topic/queue depending on JMS provider configuration.
10. ‘service-activator component’ [splitter-out-channel] is used to store log info in RDBMS layer for every first message in the group, that we get after splitting a big message. Just the same applies for the groups that consist from the one message.

The box on the screenshot below denotes utility beans in SI context:

* ‘PropertyPlaceholderConfigurer’ to provide support for a java.util.Properties inside Spring configuration file.
* ‘dbcp.BasicDataSource’ bean to provide data source to the underlying rdbms provider.
* ‘ActiveMQConnectionFactory’ to provide JMS provider connection factory to use in JMS enabled components
* ‘ActiveMQTopic’ presents dynamically created topic in Active MQ JMS Provider.

The module configuration:

# JMS Topic name

mqTopic=TOPIC.FOO

# JMS Provider host

mqHost=172.16.9.110

# local folder to pick up files to send [do not used any longer]

inputFolder=c:\\tmp\\input

# The size of the chunk to split the whole message

chunkSize=1048576

The packaging process:

Every project is bundled as maven project, the call ‘mvn clean package’ produces distributable zip bundle in the target folder. In the scripts/msdos folder there’s file run.bat to start application in test mode. In the future just the same initialization process would be completed using RS Doc delivery bundle instead of the using ‘ru.rstyle.si.main.Sender’ test case.