

Departamento de Engenharia Informática
Faculdade de Ciência e Tecnologia
Universidade de Coimbra

Application Integration with an Enterprise Service Bus

Enterprise Application Integration

Coimbra, 20th of December 2015

Flávio J. Saraiva, nº 2006128475, flavioj@student.dei.uc.pt
Mário A. Pereira, nº 1998018322, mgreis@student.dei.uc.pt

Index:

- [1. Objectives](#)
- [2. Project Overview](#)
- [3. Implementation](#)
 - [3.1. HTTP-CreateDatabase](#)
 - [3.2. Subscription services](#)
 - [3.2.1. HTTP-*](#)
 - [3.2.2. HTTP-Subscribe](#)
 - [3.2.3. HTTP-ActivateSubscription](#)
 - [3.2.4. HTTP-Unsubscribe](#)
 - [3.2.5. HTTP-ConfirmUnsubscribe](#)
 - [3.2.6. HTTP-EmailSubscriptions](#)
 - [3.2.7. SOAP-SubscriptionService](#)
 - [3.2.8. Quartz-RemoveInactiveSubscriptions](#)
 - [3.2.9. Subscribe](#)
 - [3.2.10. Activate-Subscription](#)
 - [3.2.11. Unsubscribe](#)
 - [3.2.12. Confirm-Unsubscribe](#)
 - [3.2.13. Email-Subscriptions](#)
 - [3.2.14. Get-HTTP-Params](#)
 - [3.2.15. Describe-Subscription](#)
 - [3.2.16. Digest-Subscription](#)
 - [3.2.17. Object-To-Subscription](#)
 - [3.3. Send Email](#)
 - [3.3.1 Send-Mail](#)
 - [3.3.2. Send-Mail-DEI](#)
 - [3.3.3. Send-Mail-GMAIL](#)
 - [3.4. Load Smartphones](#)
 - [3.4.1. Common-LoadSmartphones](#)
 - [3.4.2. Directory-LoadSmartphones](#)
 - [3.4.3. SOAP-LoadSmartphones](#)
 - [3.5. SendDigest](#)
 - [3.6. twitter](#)
 - [3.7. SOAP-Statistics](#)
 - [3.8. Subscription clients:](#)
 - [3.8.2. SubscriptionSOAPClient](#)
 - [3.8.1. Static HTML Page](#)
 - [3.9. WSLoadXMLClient](#)
 - [3.10. Crawler](#)
 - [3.11. WsStatisticsClient](#)
- [4. Discussion of Results](#)
- [5. Conclusions](#)

1. Objectives

The main goal of this project is to develop a system based on the very popular Enterprise Service Bus using Mule ESB and integration framework..

In order to achieve such a goal we used Anypoint Studio, an eclipse-based editor that allows us to develop such system using a Graphic User Interface or by editing an XML file.

Using such a framework we were able to develop several event based flows that allow us to build the proposed services with fully integration of external components like a database, Web Service and browser clients.

2. Project Overview

Based on the proposed objectives we were asked to develop an application that processed and managed smartphone information that was fed to the system via XML. This information should be made available to clients via a subscription mechanism.

In order to subscribe the service the clients should be able to fill in a form that would be made available through a static web page or a console client. After submitting the subscription, the service should process it and send a confirmation email to the subscriber. After accessing an hyperlink present in the confirmation email the subscription should become permanent. The system should also allow an unsubscription service that should allow users to unsubscribe the service. This subscription service should be provided via a static webpage or a SOAP Web Service that would allow a console client to interact with the services. All data relative to subscriptions was permanently stored in an external mysql database.

Since the system should rely on information obtained via XML files or strings, there should be a service that allows for the service to be fed such information. In this particular system, two variants were implemented. The first service should be linked to a file directory. When a XML file was copied into the directory, the system consumed that file and, after the XML contained it was validated, it was stored in the external database. The second service was a SOAP Web Service that allows a remote client to send a XML string that was validated and stored in the database. In order for a certain smartphone to be accepted, it's monitor should be at least 10cm wide. Smartphones with smaller screens were discarded.

Once a day the subscriber should receive a daily digest with smartphone information regarding the brand and price range they gave when the service was subscribed.

In order to do that the system had a daily digest service that executes once a day at midnight, queried the database for the pertinent information for each subscription and sended it to the subscriber via email.

Finally, all the actions of subscribing the service, sending emails, twitts and updating the smartphone database should be recorded for statistical purpose. Every service in the system updates the database whenever one of these four events occurs. and sends the information to a Twitter account. Finally there should also be a SOAP Web service that allows a client to get statistics information that was returned to the client's console.

3. Implementation

All the information presented in the server console, which includes exceptions, is logged to the `logs/Project3.log` file of the directory where the application was deployed. When the project is run from the Anypoint Studio the mule application is deployed to the `.mule` directory in the workspace.

Some features, like sending mails, use the application properties, which can be found at `src/main/app/mule-app.properties`.

All ssl/tls connections use the `src/main/resource/truststore.jks` file as the default trust store and use the `src/main/resource/keystore.jks` file as the default client store and the default keystore. The key store contains a self-signed certificate for localhost and it's private key. The trust store contains the certificate chain used the `smtp.dei.uc.pt`.

3.1. HTTP-CreateDatabase

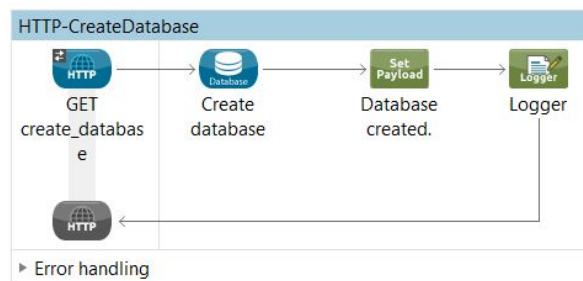


Figure 1: HTTP-CreateDatabase flow.

This is a very simple service. When a web browser accesses the web address `http://localhost:8081/create_database` the service it will send the queries in `src/main/resources/create_database.sql` to the MySQL server.

Upon completion a success message "Database created." is placed in the payload, the page content, and the result is logged. The service can be run multiple times without conflicts and can be stopped after the database is created.

isproject3.subscription	isproject3.smartphone	isproject3.statistics
<ul style="list-style-type: none">id : bigint(20)email : varchar(255)clientName : varchar(255)channel : varchar(255)favoriteBrand : varchar(255)minimumPrice : doublemaximumPrice : doubleactive : tinyint(1)created : timestamp	<ul style="list-style-type: none">ID : bigint(20)VERSION : bigint(20)TIMESTAMP : bigint(20)CRAWLER : varchar(100)URL : varchar(200)TITLE : varchar(100)PRICE : doublePROCESSOR : varchar(200)SCREENTYPE : varchar(200)SCREENSIZE : varchar(200)OTHER : varchar(200)	<ul style="list-style-type: none">ID : bigint(20)FIELD : varchar(255)NUMBEROF : bigint(20)

Figure 2: Database structure shown by phpMyAdmin 4.3.11.

The service creates the tables shown in Figure 2 that don't exist and the statistics table is filled with the fields "UPDATES", "SMARTPHONES", "EMAILS" and "TWITTS" that don't exist, with NUMBEROF set to 0.

3.2. Subscription services

These services are at the center of the system. The users can receive daily digests from the system with subscriptions.

3.2.1. HTTP-*

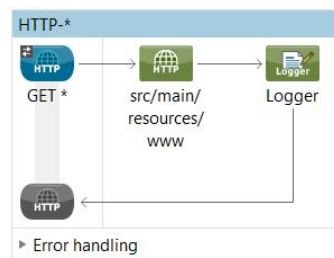


Figure 3: HTTP-* flow.

This flow waits for any request not handled by the other HTTP flows and tries to satisfy the requested path with the files in the `src/main/resources/www` folder. When a folder is requested, it provides the `index.html` file of that folder.

3.2.2. HTTP-Subscribe

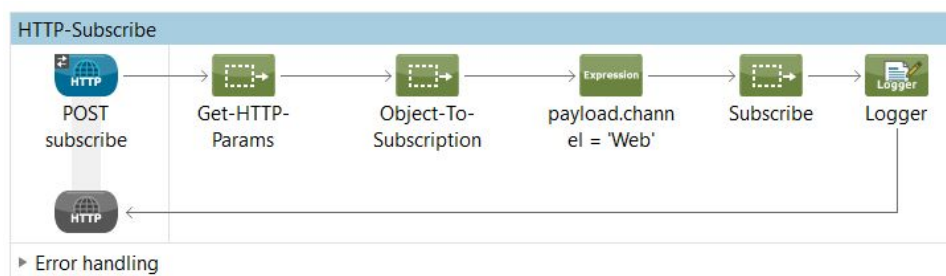


Figure 4: HTTP-Subscribe flow.

This flow allows clients to create a subscription to our system via a HTTP POST request sent from the static web page to `http://localhost:8081/subscribe`. The `Get-HTTP-Params` reference obtains the parameters sent in the request, the `Object-To-Subscription` reference turns the parameters into a `Subscription` class object, the `"payload.channel = 'Web'"` expression adds the request channel to the `Subscription` object, the `Subscribe` takes care of the subscription creation process and finally the resulting payload is logged and sent back to the client as the web page content.

A mail will be sent to activate the subscription. If the subscription is not activated within one hour it will be forgotten.

3.2.3. HTTP-ActivateSubscription

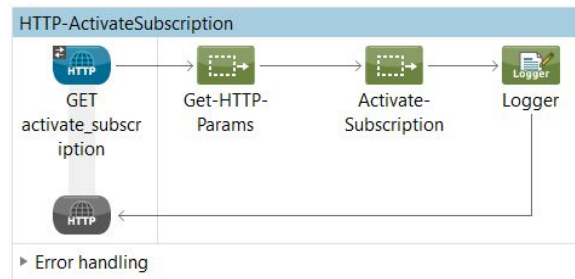


Figure 5: HTTP-ActivateSubscription flow.

This flow allows clients to activate their subscription to our system via an HTTP GET request sent from a hyperlink to http://localhost:8081/activate_subscription in the activation mail that was sent during the subscription process. The Get-HTTP-Params reference obtains the parameters sent in the request, the Activate-Subscription reference takes care of the subscription activation process and finally the resulting payload is logged and sent back to the client as the web page content.

3.2.4. HTTP-Unsubscribe

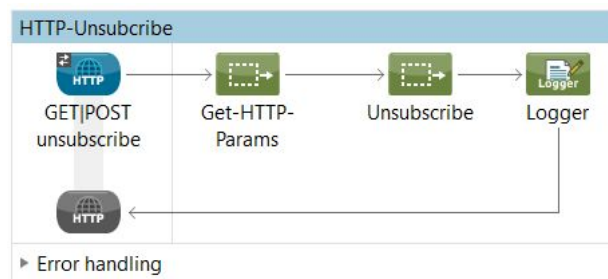


Figure 6: HTTP-Unsubscribe flow.

This flow allows clients to terminate their subscription to our system via a HTTP GET request sent from a hyperlink to <http://localhost:8081/unsubscribe> in a mail or an HTTP POST request from the static web page. The Get-HTTP-Params reference obtains the parameters sent in the request, the Unsubscribe reference takes care of the subscription termination request process and finally the resulting payload is logged and sent back to the client as the web page content.

A mail will be sent to confirm the unsubscribe attempt.

3.2.5. HTTP-ConfirmUnsubscribe

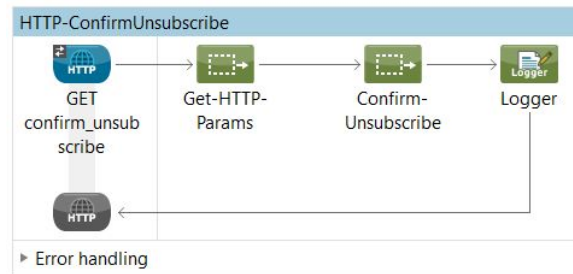


Figure 7: HTTP-ConfirmUnsubscribe flow.

This flow allows clients to confirm the termination their subscription to our system via a HTTP GET request sent from a hyperlink to `http://localhost:8081/confirm_unsubscribe` in the confirmation mail. The `Get-HTTP-Params` reference obtains the parameters sent in the request, the `Confirm-Unsubscribe` reference takes care of the subscription termination process and finally the resulting payload is logged and sent back to the client as the web page content.

3.2.6. HTTP-EmailSubscriptions



Figure 8: HTTP-EmailSubscriptions flow.

This flow will send the list of subscription to the target email via a HTTP POST request sent from the static web page to `http://localhost:8081/email_subscriptions`.

The mail will only be sent if the email has at least one subscription to avoid sending spam.

3.2.7. SOAP-SubscriptionService

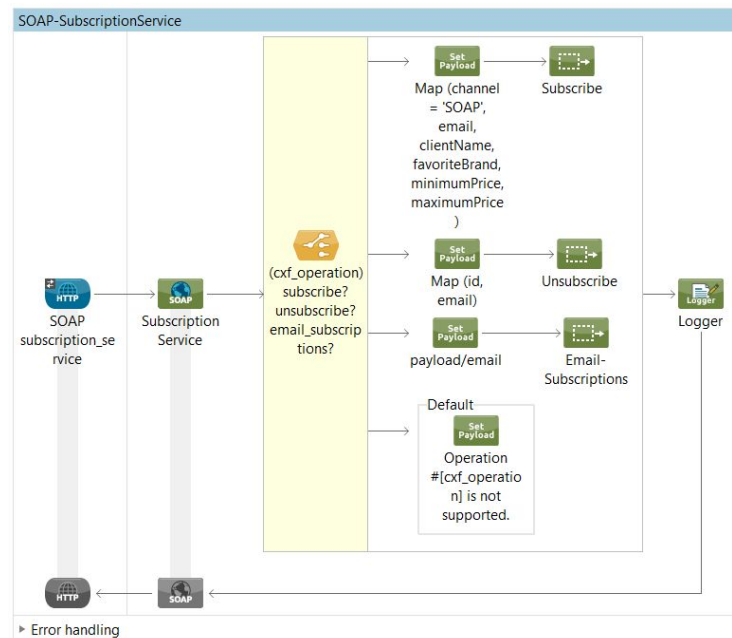


Figure 9: SOAP-SubscriptionService flow.

This flow implements the SOAP subscription web service described by the interface `is.project3.subscriptions.SubscriptionService` and is available at `http://localhost:8081/subscription_service`. This interface allows for the remote method invocation of several services in the server: `subscribe`, `unsubscribe`, and `send a mail with the current subscriptions to be sent to the user`. Which is equivalent to the flows `HTTP-Subscribe`, `HTTP-Unsubscribe` and `HTTP-EmailSubscriptions`.

After receiving a request from the SOAP client it chooses what to do based on `cxf_operation`, which contains the name of the SOAP operation. It then transforms the operation arguments to the format expected by the sub flow and executes the sub flow. If the operation is not recognized it simply returns the string "Operation #[`cxf_operation`] is not supported.". After the operation is performed the resulting string is logged and sent back to the SOAP client.

3.2.8. Quartz-RemoveInactiveSubscriptions

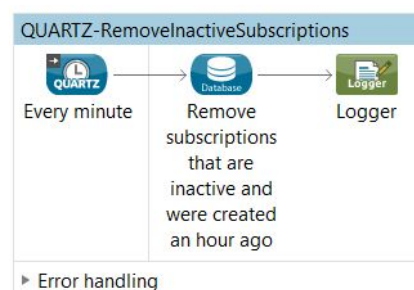


Figure 10: Quartz-RemoveInactiveSubscriptions flow.

This flow is activated every minute by the quartz component and performs a query that removes inactive subscriptions which were created at least one hour ago. This flow ensures that subscription created with a bad email address will be forgotten.

3.2.9. Subscribe

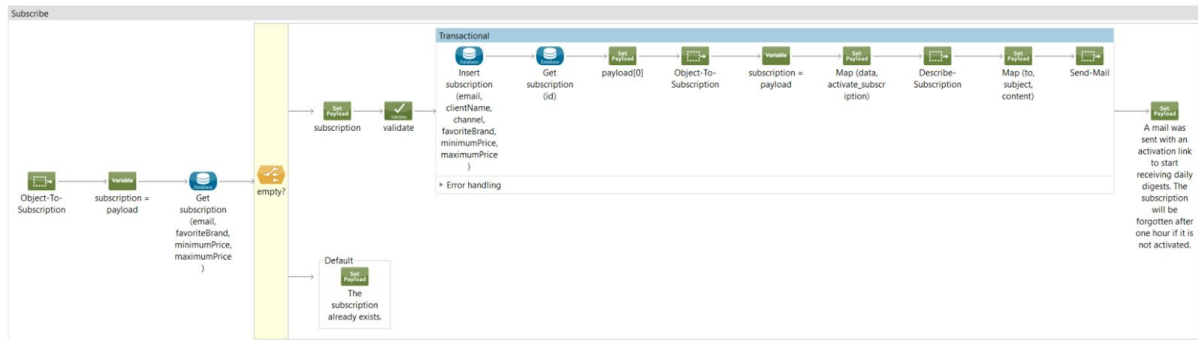


Figure 11: Subscribe subflow.

This subflow processes a subscribe request. It is referenced by the HTTP-Subscribe flow and by the SOAP-SubscriptionService flow.

Upon receiving the payload it checks if this subscription already exists in the database by comparing the email, favorite brand, minimum price and maximum price. If it already exists it simply returns the string “The subscription already exists.”. If the subscription does not exist it checks if the data in the subscription is valid, then it starts a transaction, inserts the subscription into the database, gets the inserted subscription back from the database, converts it to a Subscription class object and stores it in the subscription variable. Finally it builds an activation mail with the subscription description and the activation link, sends the mail, terminates the transaction and returns the string “A mail was sent with an activation link to start receiving daily digests. The subscription will be forgotten after one hour if it is not activated.”.

The returned string, placed in the payload, will be the web page content or the returned value of the main flow.

3.2.10. Activate-Subscription

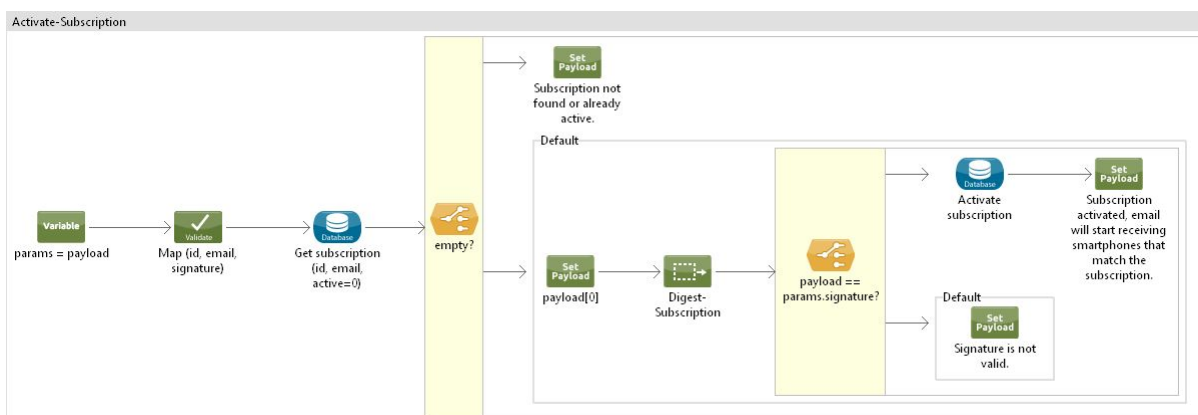


Figure 12: Activate-Subscription subflow.

This subflow activates a subscription. It is referenced by the HTTP-ActivateSubscription flow.

It starts by storing the current payload in the params variable. Then, upon validating the information received it retrieves a matching subscription that isn't active from the database. If no subscription is found it simply returns the string "Subscription not found or already active.". If a matching subscription is found it constructs the subscription signature and compares it against the signature received from the client. If the signature does not match it returns the string "Signature is not valid.". If the signature matches it updates the active field of the subscription in the database and returns the string "Subscription activated, email will start receiving smartphones that match the subscription.".

The returned string, placed in the payload, will be the web page content of the main flow.

3.2.11. Unsubscribe

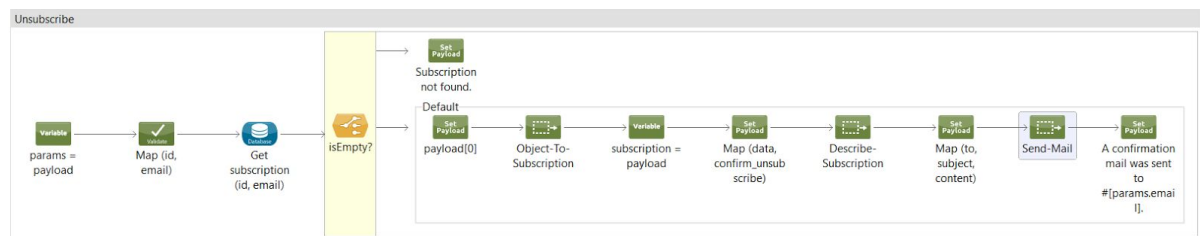


Figure 13:Unsubscribe subflow.

This subflow processes an unsubscribe request. It is referenced by the HTTP-Unsubscribe flow and by the SOAP-SubscriptionService flow.

It starts by storing the current payload in the params variable. Then, upon validating the information received it retrieves the subscription from the database. If no subscription is found it simply returns "Subscription not found.". If the subscription is found it converts it to a Subscription class object and stores it in the subscription variable. Finally it builds a confirmation mail with the subscription description and the confirmation link, sends the mail and returns the string "A confirmation mail was sent to #[params.email].".

The returned string, placed in the payload, will be the web page content or the returned value of the main flow.

3.2.12. Confirm-Unsubscribe

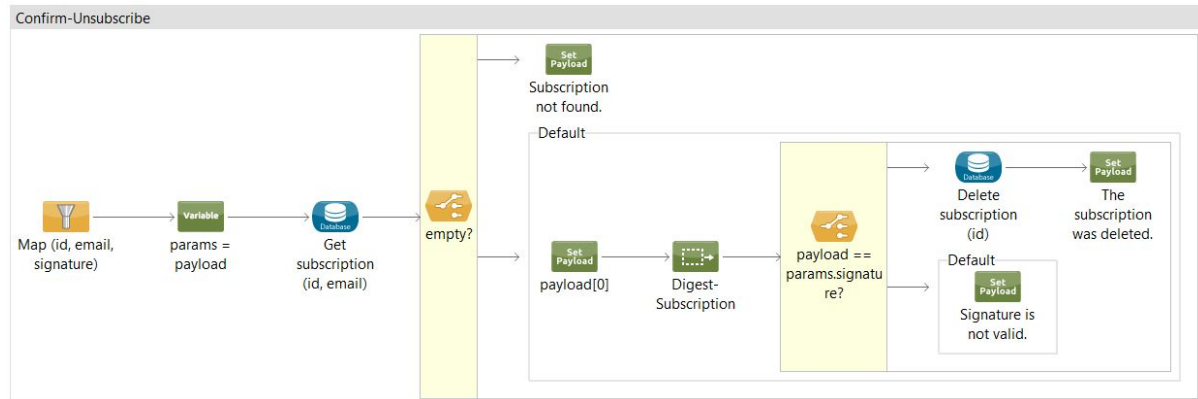


Figure 14: Confirm-Unsubscribe

This subflow terminates a subscription. It is referenced by the HTTP-ConfirmUnsubscribe flow.

It starts by checking for nulls in the current payload. Then, upon validating the information received it retrieves the subscription from the database. If no subscription is found it simply returns “Subscription not found.”. If the subscription exists it constructs the subscription signature and compares it against the signature received from the client. If the signature does not match it returns the string “Signature is not valid.”. If the signature matches it deletes the subscription from the database and returns the string “The subscription was deleted.”.

The returned string, placed in the payload, will be the web page content of the main flow.

3.2.13. Email-Subscriptions

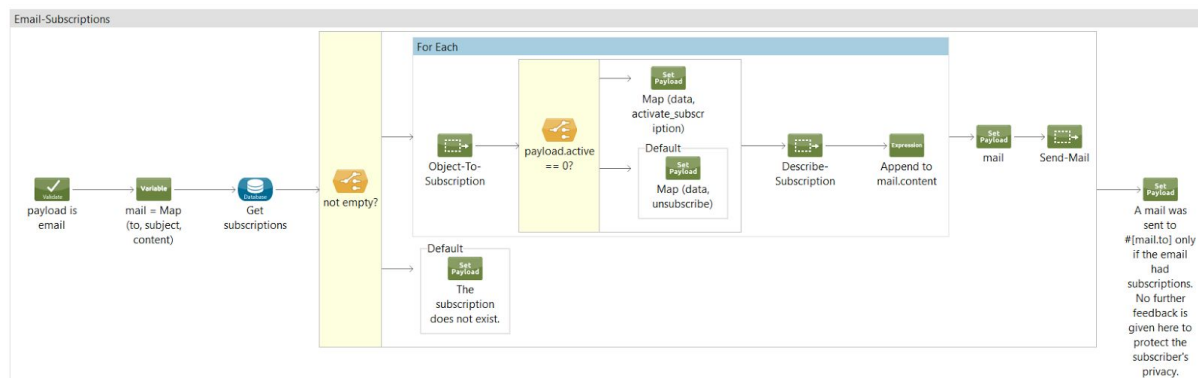


Figure 15: Email-Subscriptions subflow.

This subflow is referenced by the HTTP-EmailSubscriptions flow and by the SOAP-SubscriptionService flow.

It checks if a given email has any active subscriptions. If it does it sends a mail to the user containing the description of all its subscriptions. Inactive subscription include the activation link and active subscriptions include the unsubscribe link. Regardless of the number of subscriptions found, it always returns the string “A mail was sent to #[mail.to] only

if the email had subscriptions. No further feedback is given here to protect the subscriber's privacy.”.

The returned string, placed in the payload, will be the web page content or the returned value of the main flow.

3.2.14. Get-HTTP-Params

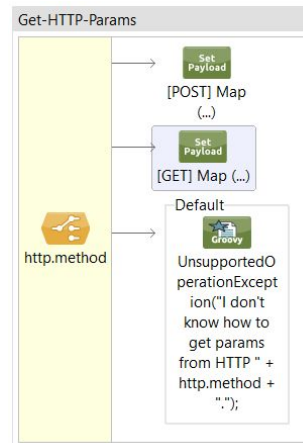


Figure 16: Get-HTTP-Params subflow.

This subflow checks what type of HTTP request was used to trigger the main flow and returns the query parameters in the inbound properties for HTTP GET or the parameters in the payload for HTTP POST.

If the request is neither a GET or a POST the system throws an exception.

3.2.15. Describe-Subscription

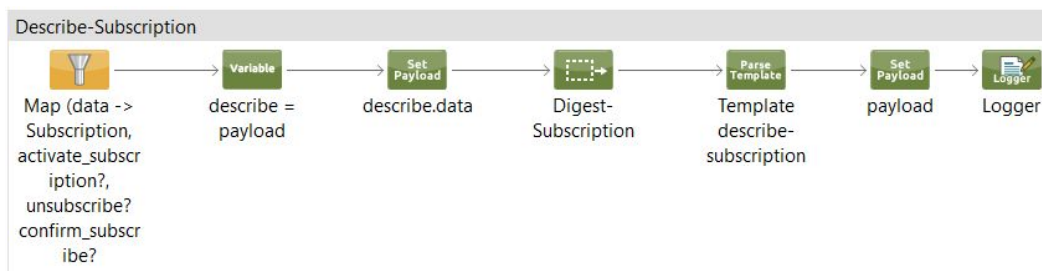


Figure 17: Describe-Subscription subflow.

This subflow verifies that it received a map with a Subscription class object in the data field, then it constructs the subscription signature and builds the description of the subscription from the describe-subscription.txt template file, located in the src/main/resources/template directory.

The map has optional boolean fields that indicate which links should be included in the description: activate_subscription, unsubscribe and confirm_subscription.

3.2.16. Digest-Subscription

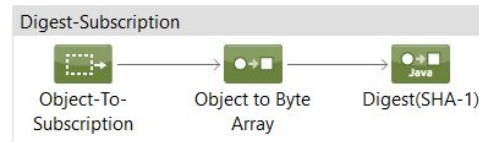


Figure 18: Digest-Subscription subflow.

This subflow generates the signature of a subscription. It converts the payload to a Subscription class object, then converts the Subscription class object to a byte array and digests the byte array into SHA-1 hash. The hash is represented by a hex string.

3.2.17. Object-To-Subscription

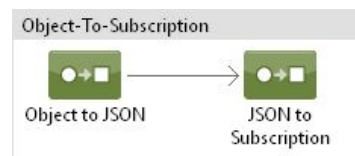


Figure 19: Object-To-Subscription subflow.

This subflow converts an object, usually a map, to a Subscription class object with standard transform components by temporarily converting the data to JSON.

3.3. Send Email

We support sending mail through the DEI smtp server with a SSL connection and through gmail with an unprotected connection.

3.3.1 Send-Mail

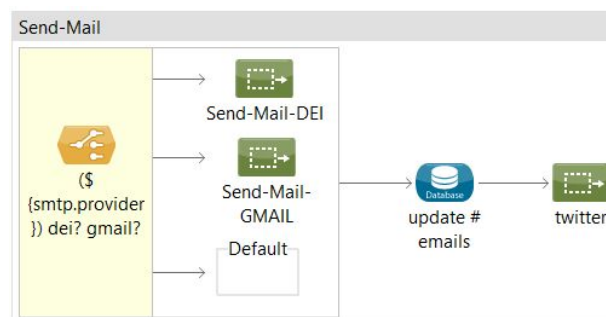


Figure 20: Send-Mail subflow.

This subflow is referenced is responsible for selecting what SMTP service should be used to send the mail. Upon being sent the statistics table is updated and the twitter flow is called to report the change.

The choice is based on the `smtp.provider` application property, which can be either “dei” or “gmail”.

3.3.2. Send-Mail-DEI

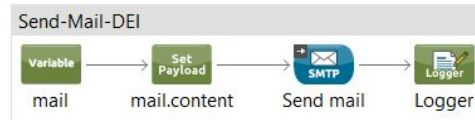


Figure 21: Send-Mai-DEI subflow.

This flow sends a mail through the `smtp.dei.uc.pt` server using the SSL port 465. The event is then logged. The from email

Some fields are filled from the application properties. The from field is filled with `${smtp.from}`, the username field is filled with `${smtp.user}` and the password field is filled with `${smtp.password}`.

3.3.3. Send-Mail-GMAIL

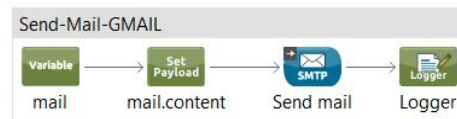


Figure 22: Send-Mail-GMAIL subflow.

This flow sets the message and sends it through the `smtp.gmail.com` server using the port 587 (unencrypted). The event is then logged.

3.4. Load Smartphones

3.4.1. Common-LoadSmartphones

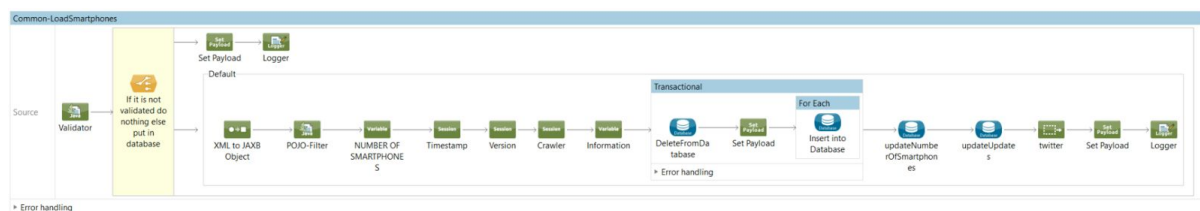


Figure 23: Common-LoadSmartphones flow.

This flow is referenced by both the directory and SOAP based `loadSmartphones` service.

Upon being referenced it validates the XML string against an XSD schema. If it passes the process continues, else it sets a failure message in the payload and logs the result.

After being validated the string is transformed into a JAXB Object that is filtered to remove all the smartphones that have a screen smaller than 10cm. All the smartphones that fit the requisite are then inserted into the MySQL database smartphone table that has been wiped before insertion.

Finally the Statistics table is updated regarding the number of smartphones and the number of updates to the smartphones table. Finally the success message is put in the payload and a log entry is written.

3.4.2. Directory-LoadSmartphones

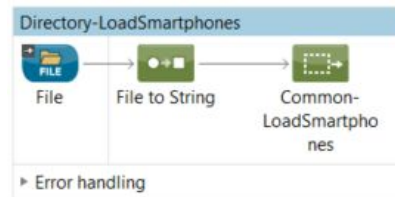


Figure 24: Directory-LoadSmartphones flow.

This simple flow is activated when a file is put on a given directory. It Consumes the file, turns it into a string a passes control to the Common-LoadSmartphones flow through a reference.

3.4.3. SOAP-LoadSmartphones

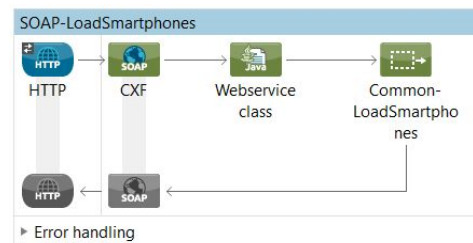


Figure 25: SOAP-LoadSmartphones flow.

This simple flow allows the system to receive a XML string containing smartphone information via a Web Service using SOAP protocol.

A client can use this web service to upload XML to the server by simply using a remote method invocation and all the communication aspects are handled by the lower communication layers in a transparent way.

In this particular service the client invokes a remote method loadXML() passing a String containing XML as a parameter and receives another String containing a success or failure message.

3.5. SendDigest

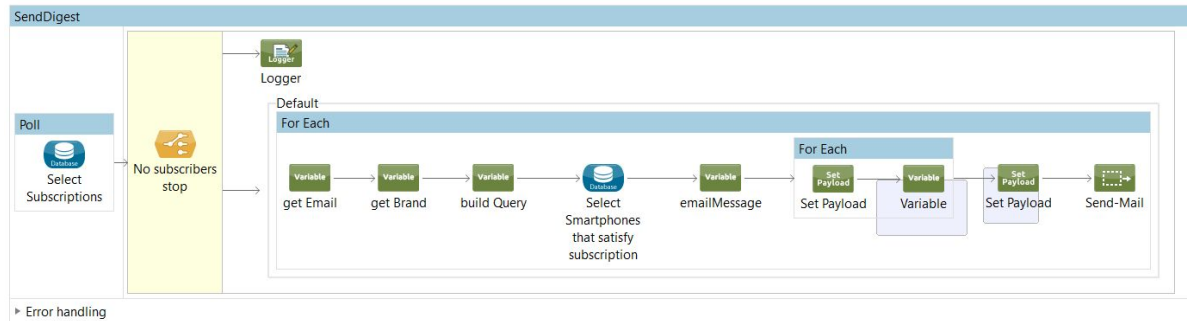


Figure 26: SendDigest flow.

This flow allow for the server to send an email digest to subscribed users once a day. In this particular configuration it sends a digest right away when it starts and continues sending a message once a day at the same hour. Even though the project specified for a mail to be sent once a day we opted to send a mail right away to demonstrate this functionality.

In order to send the digest once a day at midnight we could just insert a cron string in the poll component with the expression "0 0 * * *" that would allow for a message to be sent at exactly 0:00 every day the system was active.

Upon being activated by the Poll component, the service queries the database and receives a list of active subscriptions. If there are no active subscriptions the system stops and logs the event. If there are active subscriptions the system stores important information regarding the subscription like the Email and the chosen brand and builds a query to select from the database the smartphones that fit criteria of brand and price present in the subscription. Then it builds the appropriate message and uses flow reference to call the Send-Mail flow so that the digest can be sent.

Even if there are no Smartphones that fit the criteria imposed the email is sent anyway as a reminder to the user that he has a subscription active in the system. As the available smartphones may vary it is possible for the subscribed user to receive different emails every day.

3.6. twitter

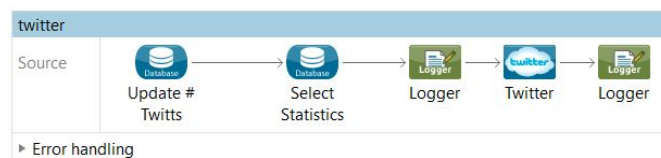


Figure 27: twitter flow.

The twitter flow is referenced in several other flows each time there is an update to the smartphones table or an email is sent.

This flow gets from the MySQL database all the data contained in the statistics table. Then it builds a message and updates a Twitter account status with it. Finally it logs the action.

3.7. SOAP-Statistics

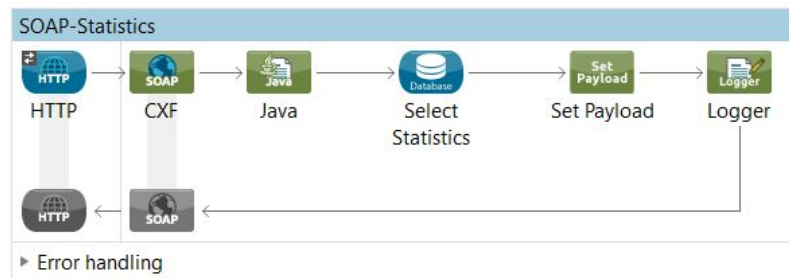


Figure 28: SOAP-Statistics flow.

Beside consulting the twitter account, users can check the statistics of the system by requesting it to the Web Service using SOAP represented by this flow.

Upon receiving a request the flow requests all data from the Statistics table in the MySQL database, sets it in a message, logs the action and returns the data to the client.

3.8. Subscription clients:

We were asked to make subscribe and unsubscribe available in the following two ways: a SOAP client with a text-based user interface and a static web page. For user convenience the clients can also send the subscription list of an email to that email, but will only send it if there is at least one subscription to avoid sending spam.

3.8.2. SubscriptionSOAPClient

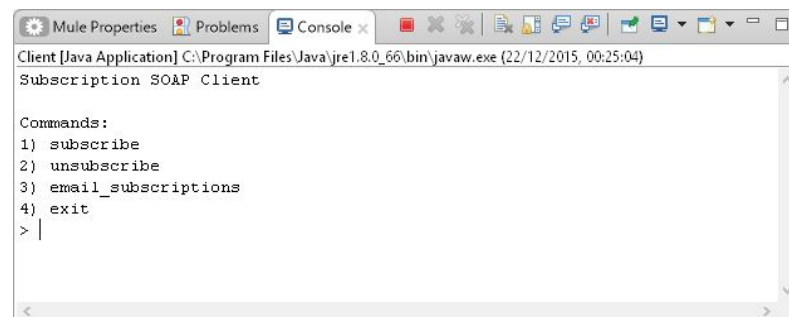
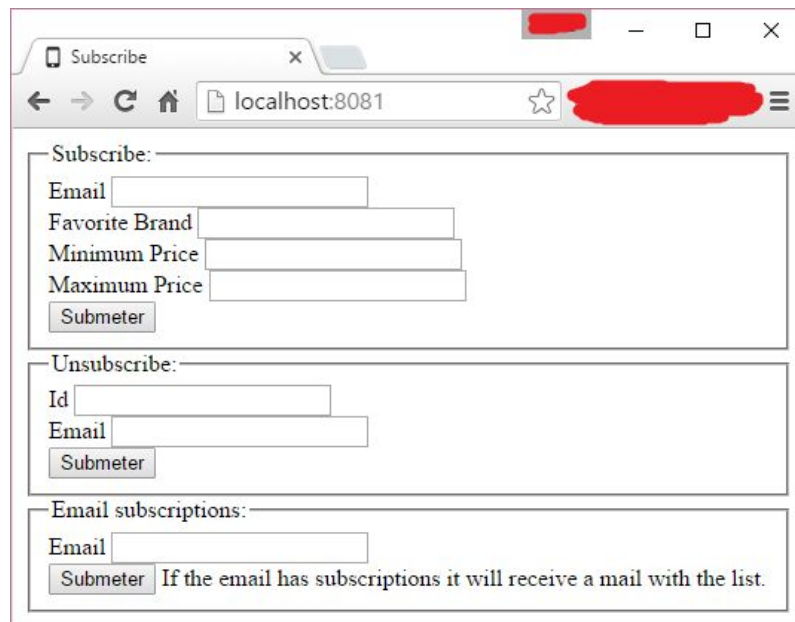


Figure 29: SubscriptionSOAPClient running inside Anypoint Studio.

This small program allows users to manage their subscriptions using the text console. This application makes requests to the SOAP subscription Web Service available at http://localhost:8081/subscription_service, which replies with a string describing the result of the request.

3.8.1. Static HTML Page



The screenshot shows a web browser window with the title 'Subscribe'. The address bar shows 'localhost:8081'. The page content is as follows:

Subscribe:

Email

Favorite Brand

Minimum Price

Maximum Price

Unsubscribe:

Id

Email

Email subscriptions:

Email

If the email has subscriptions it will receive a mail with the list.

Figure 30: Static HTML Page opened in chrome.

This static web page, the Static HTML Page client, allows users to manage their subscriptions from any web browser. It makes HTTP POST requests to the server and can be accessed at <http://localhost:8081/>.

3.9. WSLoadXMLClient

This small application allows users to upload an XML file containing smartphone information using the SOAP LoadXML WebService.

3.10. Crawler

The web crawler from project 1 has been modified. Instead of publishing the XML String in a JMS Topic it uses the SOAP LoadXML WebService to upload it to the system.

3.11. WsStatisticsClient

This small program allows users to check the statistics of the system by requesting it to the Web Service SOAP-Statistics. When this small application is run in the console it returns the relevant information to the screen.

4. Discussion of Results

Using the Mule ESB and integration framework we were able to fully implement the project as well as some extra features.

After a few initial days learning how the tool works it became clear how powerful Mule ESB architecture is. With it it was possible to fully integrate very distinct applications in a short period of time. The use of the Anypoint Studio editor GUI and its pre constructed components further allowed us to develop our system in a very intuitive way.

The event driven flows and the payload manipulation tools hide most of the complexity of the communication layer and allowed us to concentrate our efforts in what transformations should occur to our data and what the behavior of each flow should be.

Were we to fully develop the system services using Java, it would have been impossible to deploy such a complete system in such a short period of time.

5. Conclusions

The Mule ESB was an invaluable tool for implementing such a complex system in a short period of time.

By concentrating our efforts in message manipulation and the flow behavior, and by using a series of pre constructed elements we were able to develop all the services and a few extra features .

With the project it became clear how the ESB Architecture is highly desirable when there is a need to develop a system that has to integrate so many different components and communicate with clients via such different technologies.

All the Project Objectives were achieved.