Grenoble INP – ENSIMAG
Ecole Nationale Superieure dInformatique et de Mathematiques Appliquees

Second year internship report

At Solystic SAS

Implementing a new architecture for a software suite integrated to the Information Systems inherent in the field of object sorting

RAMARIJAONA Maeva
2nd year – ISI Specialization

June 1st 2017 - September 13th 2017 (14 weeks)

Solystic SAS
21 rue de Chony
BP 102
26501 BOURG LES VALENCE CEDEX

Internship supervisor
CROUZET Lionel
Ensimag Reviewer
?

Contents

1 Introduction

1.1 Company Overview

Solysitc is a French company that is a wholly owned subsidiary of Northrop Grumman and specializes in mail sorting. It provides sorting, coding and reading systems used in mail sorting centers all around the world, softwares to help program and monitor these systems and assistance and maintenance of the systems in sorting sites. With more than 800 patents applications filed and 7% of yearly spendings dedicated to R&D, Solystic tries to accomodate the multiple needs of their diverse clientèle by constantly trying to improve the services they make available.

With clients in 28 countries and an average annual revenue of $120M , Solystic is the leader company in the sector of mail sorting.

1.2 Object of the mission : the SoMoS II application

My internship was centered around modifying an already existing application called SoMoS II, developed in 2016 It is written in PHP and uses the Symfony framework, and usually deployed on an Apache server. This application was originally intended for use by BPost which is the Belgian postal service.

This application consists of three main parts : Administration, Data Management and Scenario Management. The Administration part is used to manage the production resources, which means the sorting systems, sorting sites and sorting parameters, as well as the user rights. The Data management part helps manage data from 2 different databases, AVCS and ROMA. Finally the Scenario part allows a user to create, view, edit, run or delete a scenario in order to simulate or predict the sorting and distribution of mail according to the production resources information provided by the other two parts.

2 Dividing an application in services connected by a bus

2.1 A global architecture issue more than a problem with the particular application

The SoMoS II application offers a range of useful functionalities and is practical for a client on its own. However, it is when associated with other applications that the current architecture shows some flaws. For instance, most of the functionalities seen in the Administration part are useful in other applications, yet those applications have no means to share these features with SoMoS II. That means they each have to have an implementation of these functionalities, and this poses an issue when an update is needed on any of these features, as all the applications have to be updated. This architecture also makes it difficult to upscale the services. Yet, for Solysitc's clients, upscaling is essential as sorting sites may change capacity over time and some

house hundreds of systems that might each send thousands of requests each to these different services.

Building such monolithic and inter independent applications may ensure a sense of stability, indeed if an application crashes, the others are unaffected and a given functionality is still available somewhere. Yet, this redundancy makes for complex and voluminous applications that are energy and time consuming to deploy and use. In an industry such as mail sorting where time is a crucial parameter, any improvement that may cut costs and/or treatment time is to be taken. In this regard however, Solystic seems slightly behind their main competitor, and that is why, in order to increase the efficiency of their software and gain new markets, they started considering a new approach.

2.2 Enterprise Application Integration

To work on these problems my supervisor asked me to do some research on Enterprise Integration.

Starting as soon as the beginning of the millennium, companies started finding out the type of issues mentioned earlier in their software architecture. To answer this need to increase efficiency, a new type of architecture in companies was adopted : Enterprise Application Integration (EAI). This consists in building a middleware that connects different and potentially heterogeneous applications, by implementing Enterprise Integration Patterns (EIP).

2.2.1 The state of the art : Two main trends in EAI

The most common implementation of EAI nowadays is Service Oriented Architecture (SOA). This architecture consists in presenting each application as a service that can receive, send and process only certain types of messages. These messages are sent via standardized protocols, often SOAP. The services communicate through a piece of architecture called and Enterprise Service Bus (ESB) which then connects to the services through adapters. This is particularly useful for making applications that come from different vendors cooperate. The use of EIP allows the ESB to not only transfer the messages but also format them in order to be adequate for whichever service they are being sent to. The main challenge of this model is to build the ESB in order to not make it a bottleneck where messages pile up and slow down the whole system.

Besides SOA a new trend of integration has surfaced recently : microservices. In this architecture, applications are divided into even smaller services, often implementing only one function, and communicating directly with each other, most of the time using RESTful APIs. This very fine grained distinction between services make the applications even more scalable and easy to maintain. However, if a company is starting from a system of monolithic complex applications, the transition might be more difficult as this type of architecture forces a wholly new type of team organization, according to Conway's law : "Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.". This approach is sometimes combined with the former to create an architecture where very small services communicate through a very basic bus following the

principle of ”dumb pipes and smart endpoints” .

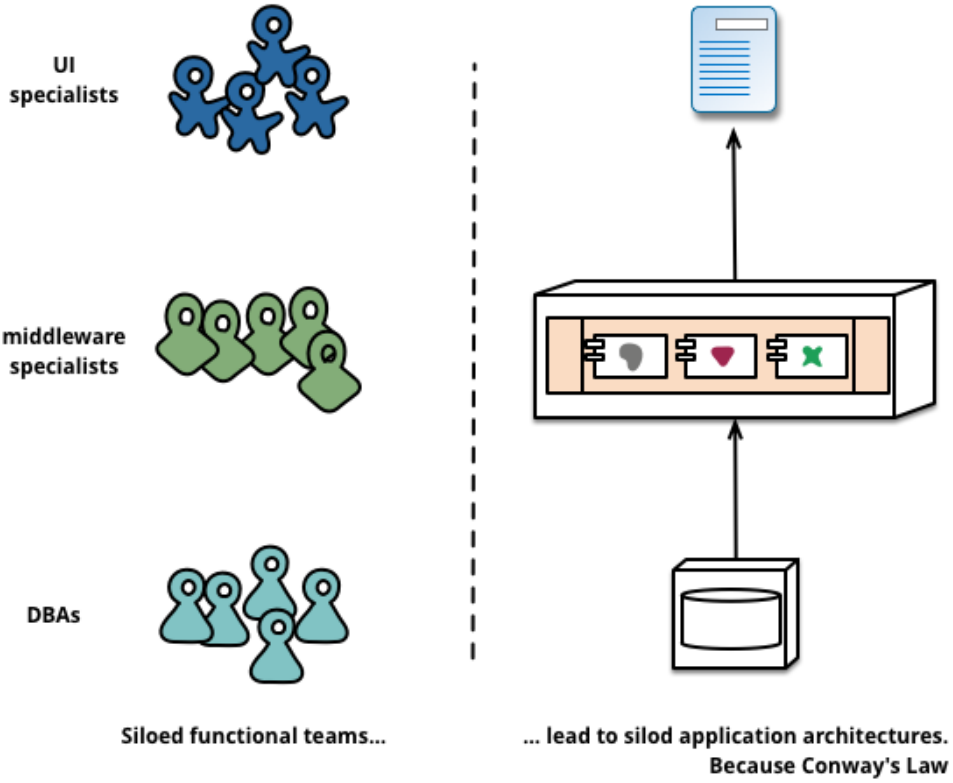


Figure 1: An illustration of Conway’s Law

This means that the bus is used merely as a message store but does not do any routing or other modification on the messages like a ESB in an SOA would. This helps combine the asynchronism of the SOA with the extreme modularity of a microservice architecture.

2.2.2 Choosing a model

As stated earlier, each model has its advantages and inconveniences. However, using the SOA was often pointed out as more ”beginner friendly” than jumping straight to the microservices model.

We decided to follow the multiple pieces of advice on this subject and start transitioning by implementing an SOA architecture. Indeed it seemed more logical to only divide an application in a couple of services and focus on implementing the communication of the two parts with a bus. There was already some of thought put into what services might ultimately be needed when the transition would be complete, so my mission was mostly to figure out a way to imple-

ment this transition on SoMoS II which originally contained the "Production Resources Management", "Production Planning Management" and "Delivery Planning Management" services.

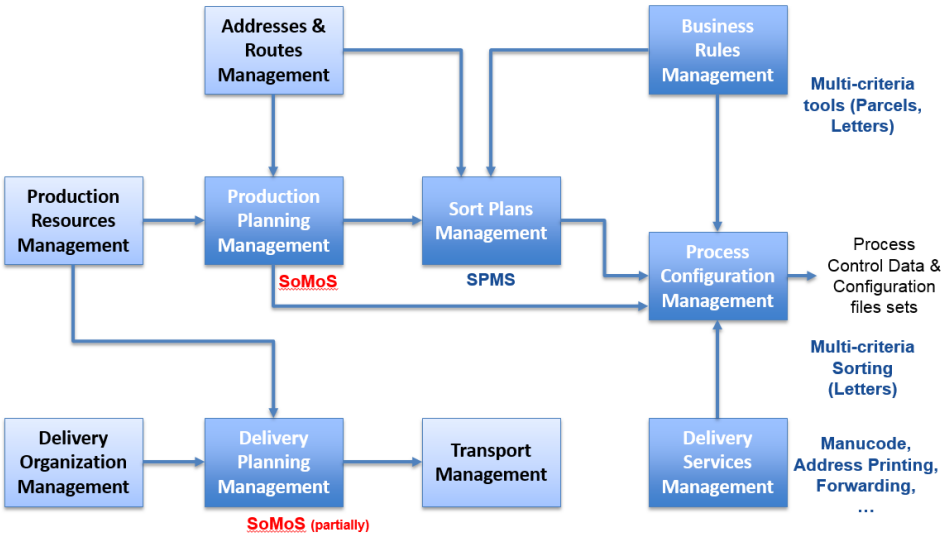


Figure 2: The different services envisionned by Solystic

2.3 Choosing a product

With EAI being such a trendy subject among all companies, there exists a plethora of different ESB or similar integration solutions to choose from to implement the expected change in architecture. I was however provided with a few constraints. First, the solution found would have to preferably be open source, then it should include a compatibility with a MongoDB database, and finally be reasonably priced and easy to use.

I was able to narrow down my choice to two products among the most popular, due mostly to their relatively easy use even without any experience with middleware or integration in general. Those two products were MuleSoft's AnyPoint Platform and WSO2 Enterprise Integrator. These two products are very similar, for instance they both offer a graphic user interface with which a user can manipulate components and create sequences that translate into XML files. They are also both available "on-location" or in the Cloud. They are also adapted for use in an SOA or a microservice type of system.

Mulesoft's product is used by many major companies, for example Netflix, which during my research has been presented as a leader in terms of microservices implementation. On the other hand, WSO2's solution is used by some government agencies, schools and medical centers around the world. I chose WSO2 Enterprise Integrator, not only because it was fully open source and free to download, but also because the first contact seemed slightly easier.

Enterprise Integrator (WSO2 EI) joins multiple functionalities. In addition with providing an ESB, it also contains tools to manage Database Services and Business Processes and offers dashboards to monitor the message activity on the ESB. A significant part of my internship

was dedicated to discovering and experimenting with this product in order to be more familiar with it and be able to write some documentation about it for future members of this integration project to use.

3 Methodology

3.1 Global architecture of the solution

The first step to the separation is to separate the "Production Resource Management" service from the Scenario part of SoMoS II and make the two parts communicate through the ESB. This step is itself divided in substeps:

1. Keep the application whole but send the communications between the PRM part and the Scenario part through the ESB

2. Separate only the classes composing the PRM service but leaving the direct data connexions in the application

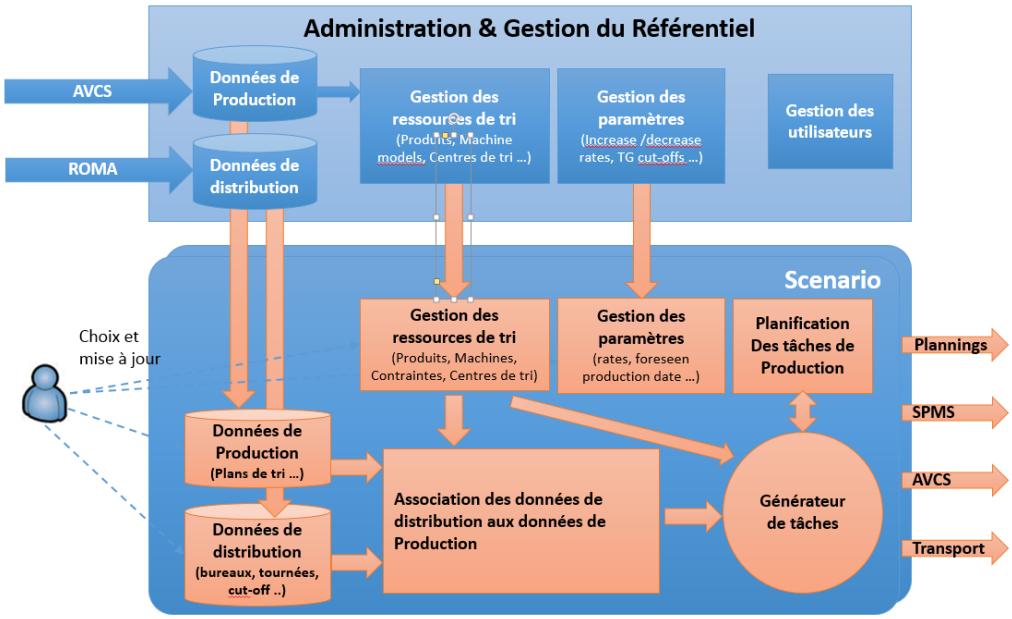


Figure 3: The basic SoMoS architecture

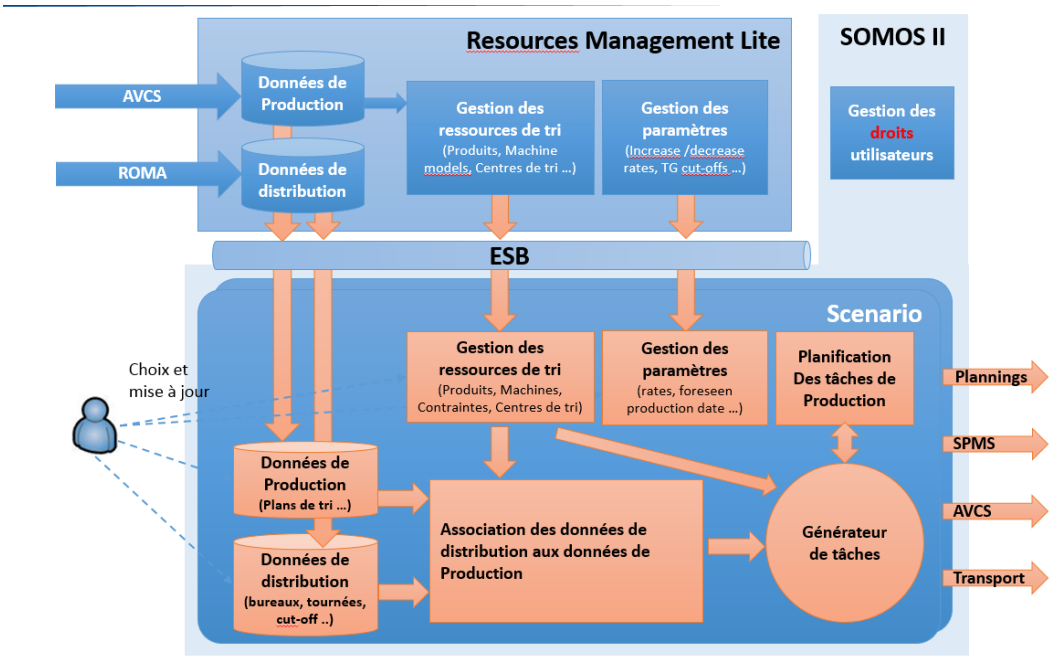


Figure 4: The direction in which the project is going

3.2 Detailed implementation

To achieve this separation, we first need to know where the link between the Production Resources Management (PRM) and the Scenario Management part is situated.

To do this I needed to closely examine the source code of the application. The application is built following a Model, View, Controller architecture. My first instict was then to examine the controllers, and most specifically the Scenario controller. There, and then in the other controllers I found which URLs the ESB needed to connect with, using the `@Route` markers. I was intending to make the communications possible via HTTP as some controller functions already produced a `Response` type result when called.

Looking through the Model classes, one PRM-to-Scenario link stood out to me in a `ScenarioManager` class. Indeed, this class contained the methods to create, view, edit and delete the scenarios and link them to the SoMoS database. The `build` function gets the Production Resource data from the AVCS and RoMA databases using methods completely equivalent to the ones used in the controller classes. In the `ScenarioManager` class I created a new function `getFromESB` that utilizes the cURL lib present with PHP 7 to send a request through the ESB to get the data using the URLs found earlier.

In the ESB, I created an API resource that forwards the GET request to the desired controller and then sends back the `Response` with a JSON message payload. The function then returns a list of objects that can be used in the `build` function derived from this JSON payload. If the payload did not exist or was not valid, the function would return an empty list. Unfortunately, every attempt to create a Scenario using `getFromESB` resulted in a redirection to the login page

and an empty Scenario even though the PRM databases were properly initialized with a set of test values.
As the Users Management part was not supposed to be affected by the project I spent little time studying it during my first examination of the source code, meaning that with the little time I had left, I would not risk modifying the user rights to access the services directly without having to go through the login step. And in the long run the same problem would have come up again as it would be unthinkable to produce an application with absolutely no credentials asked to access this type of data. To fix this issue, i tried to add a new endpoint connecting the ESB to the `login` or `login_check` page before going to the designated URL in order to pass the credentials using a `POST` request as I learned from the Symfony documentation was the procedure when a login occurred. These attempts did not work either and the messages were lost in between endpoints.

I was unable to go past this obstacle which means the separation is not complete to this day. My supervisor mentioned the possible need to shift the user rights management to a centralized login procedure, which was not in the scope of the project.

4 Testing the system and visualizing the results

4.1 A first simple application to learn how WSO2 EI works

Before going to the main subject of this project, i was given a first small application to start familiarizing myself with WSO2 EI as well as testing how much workload it could handle.

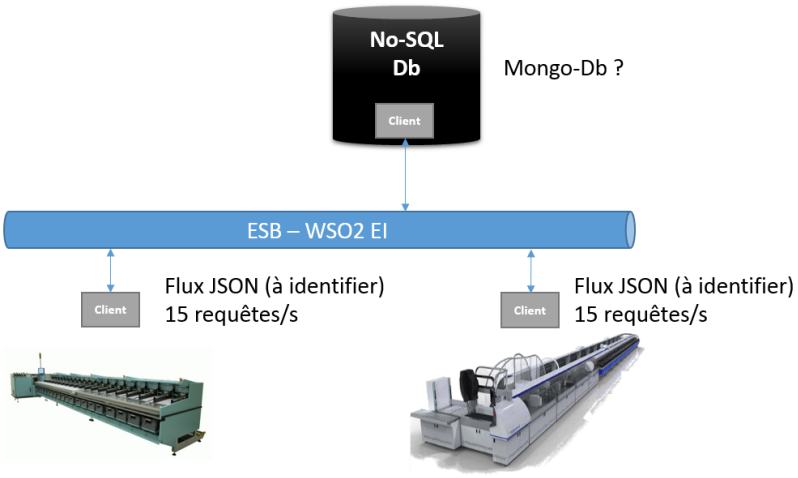


Figure 5: The WSO2 EI training project given by Solystic

A given number of sorting machines need to send data about parcels to a database. An example of the type of message produced by a machine is given in the Appendix. The only infor-

mation that needs to be saved to the database are the values of the `PracelId` and the `MachineId` fields. A full explanation on how WSO2 EI was configured to achieve this result is included in the Appendix.

To test if this a configuration works, I needed to:

- Allow tracking on the API, the proxy services, the sequence and all the endpoints
- Activate the Analytics profile
- Have a means to simulate a machine sending a message to the API

This last point is the one that varies according to which property is tested.

To test if a message sent by the machine reaches the desired endpoint, i used cURL with the verbose option, as only one message needed to be sent. After receiving an OK response via cURL, I could also verify that the message did indeed pass through all the intended services, endpoints and sequences to the database using the Analytics dashboards. From this display, I could check the changing payload and acceptance from the Database service.

When the system is correctly configured, it is time to test how much workload one entity can handle. At first i tried to set VMs as message senders, to simulate the fact that multiple machines are actively soliciting the system. This could have been possible but such a setup needed the installation of a third party load balancer and further configurations. Time constraints discouraged me from attempting to install and learn about yet again one new voluminous tool on my own.

My supervisor introduced me to a testing tool that Solystic uses regularly : Apache JMeter. After configuring the tool to send from 5 threads to the service at a rate of 15 messages per second for approximately 10 seconds at a time, I could view the results via WSO2 Analytics, this time focusing on the Requests per second panel. The 75 messages per second were handled correctly. However, if the workload was doubled, with 10 threads instead of 5 working at 15 messages per second, the ESB peaks at 85 messages processed per second even if the message processor was configured to be able to handle up to 500 messages at a time.

4.2 Trying to test the project solution attempt

After making the modifications on SoMoS II and configuring the ESB, I deployed the SoMoS application on the Apache server, activated the Analytics profile of WSO2 EI and attempted to create a Scenario using default values. After the application marked the Scenario as completed, I checked the values of the different production resources fields only to find them all at the N/A value. That means the value of the `Response` awaited by `getFromESB` was either non-existent or invalid. When checking WSO2 Analytics, the message was marked as successfully transmitted but it was not possible for me to see the payload of the response sent by the service. Trying to contact the ESB with a simple in-terminal cURL command showed me that the message was redirected towards to the login page.

After applying the "patch" described earlier, i.e. trying to pass the message to the login page BEFORE changing it in the ESB and sending it to the service, I ran the same tests in the same

order. Attempting to create a Scenario via the SoMoS II application still prompted an empty Scenario in the system. Checking Analytics yielded a rather surprising result: there was no trace of the passage of the message through the ESB even after checking thouroughly that tracing had been enabled on every component of the system. Running an in-terminal cURL command returned an "empty response from server" type of error.

5 Personal Record

5.1 First professional experience

This was my first internship and I was somewhat intimidated by the environment. I did work alone, as the members who worked on the SoMoS II application were situated in Bagneux and I was in Bourg-lès-Valence. My only regular contacts in the company were my supervisor and the SoMoS project manager whom I contacted via e-mail or phone whenever i had a question about the application.

This did not help with my integration in the department, even though i was present for 3 months and a half. That combined with personal problems, I could say that my personal daily experience could have been better.

5.2 Working with new tools

All the tools and environments used in this internship were new or unfamiliar to me. Firstly, working in a Windows environment was somewhat destabilizing as I was used to work at school as well as at home on Linux operating systems. Then, the concept of Enterprise Application Integration was entirely unknown to me, as I was used to thinking about software architecture on a smaller scale. Not only that, but the products I had to test (in particular WSO2 EI) were new to the company too, so any help that I needed I had to look for myself on WSO2 documentation or other sources.

Concerning SoMoS II, I was not familiar at all with the PHP language but my knowledge of object oriented programmation helped me understand the syntax and structure of the application. This sadly was not enough to help me understand the inner workings of Symfony's security system and fix the login issue I encountered.

I was also introduced to corporate working tools such as RedMine and SVN, useful for versioning and assigning missions. I did not get a lot of use of these except for the redaction of a short Wiki article explaining how to install and understand WSO2 EI through the simple training application i developed in the first half of the internship.

This lack of prior knowledge of any of the elements I used in this internship did make my work slow, as i had to review the basics and read a lot of documentation. It might be one of the reasons why the objectives of this internship were not met.

5.3 Supervising a student

During the last week, I was introduced to a first year Ensimag co-op student who will be continuing the project started during this internship and pushing further the transition to the new architecture model. Helping her get started with the different tools used in the company, or that I introduced helped me visualize better the amount of knowledge i have aggregated during this experience.

6 Conclusion

As a conclusion, even though this intership's goal was not met, it was still a tremendously enriching experience. Indeed, I gathered new knowledge and skills that i did not obtain through my school education. This experience did help me to explore beyond my comfort zone and to push my limits. I was honored to be tasked with a mission with this much responsibility even if my skills might not have lived up to the company's expectations.

Appendices

Glossary

1. **EAI:** Enterprise Application Integration. A type of architecture in which different applications are connected.
2. **SOA:** Service Oriented Architecture. A model that is part of EAI and consists in a number of services communicating through a bus.
3. **ESB:** Enterprise Service Bus. A piece of middleware that transfers, routes and sometimes modify messages from one service to another.
4. **PRM:** Production Resources Management. One of the services contained in the current SoMoS II application. It helps manage (view, edit create and delete) the sorting types, sorting sites, machines, sorting parameters and the machine characteristics.
5. **WSO2 EI:** WSO2 Enterprise Integrator. WSO2's integration solution. It includes an ESB, a Message Broker that helps manage message queues and message consumers, a Data Service Server, and an Analytics profile that is useful for monitoring the messages going through the system.

References

- *Integration of Distributed Enterprise Applications: A Survey* - Wu He, Li Da Xu (2011)
- *Microservices* - Martin Fowler (2014)
- WSO2 EI Documentation
- PHP manual
- Symfony manual

7 The Wiki I wrote on using WSO2 EI

Wiki ¶

Modifier Surveiller Renommer Supprimer Historique

1. Documentation

Les ressources qui m'ont aidé dans ce projet

1.1. WSO2 Integrator

Toute la documentation de WSO2 Enterprise Integrator:
https://docs.wso2.com/display/EI611/WSO2+Enterprise+Integrator+Documentation

En particulier,

- Le tutoriel d'installation
  https://docs.wso2.com/display/EI611/Quick+Start+Guide
- Les tutos d'intégration
  Envoyer un message à un service
  https://docs.wso2.com/display/EI611/Sending+a+Simple+Message+to+a+Service
  Envoyer un message à un service de données
  https://docs.wso2.com/display/EI611/Sending+a+Simple+Message+to+a+Datasource
  Créer un service de données
  https://docs.wso2.com/display/EI611/Exposing+MongoDB+as+a+Data+Service
- Toute la documentation sur le message broker
  https://docs.wso2.com/display/EI611/Message+Brokering

1.2. L'utilisation des messages JMS

- La doc Oracle
  https://docs.oracle.com/cd/E19798-01/821-1841/bncgv/index.html
- Des exemples proposés par WSO2
  https://docs.wso2.com/display/EI611/JMS+Usecases

https://docs.wso2.com/display/EI611/WSO2+EI+as+a+JMS+Consumer

https://www.yenlo.com/blog/wso2-use-the-wso2-message-broker-in-your-wso2-esb-services

1.3. WSDL et SOAP

La documentation officielle de WSDL
https://www.w3.org/TR/wsdl

Des exmeples de messages SOAP
https://documentation.pingidentity.com/display/PF70/SOAP+Request+and+Response+Example

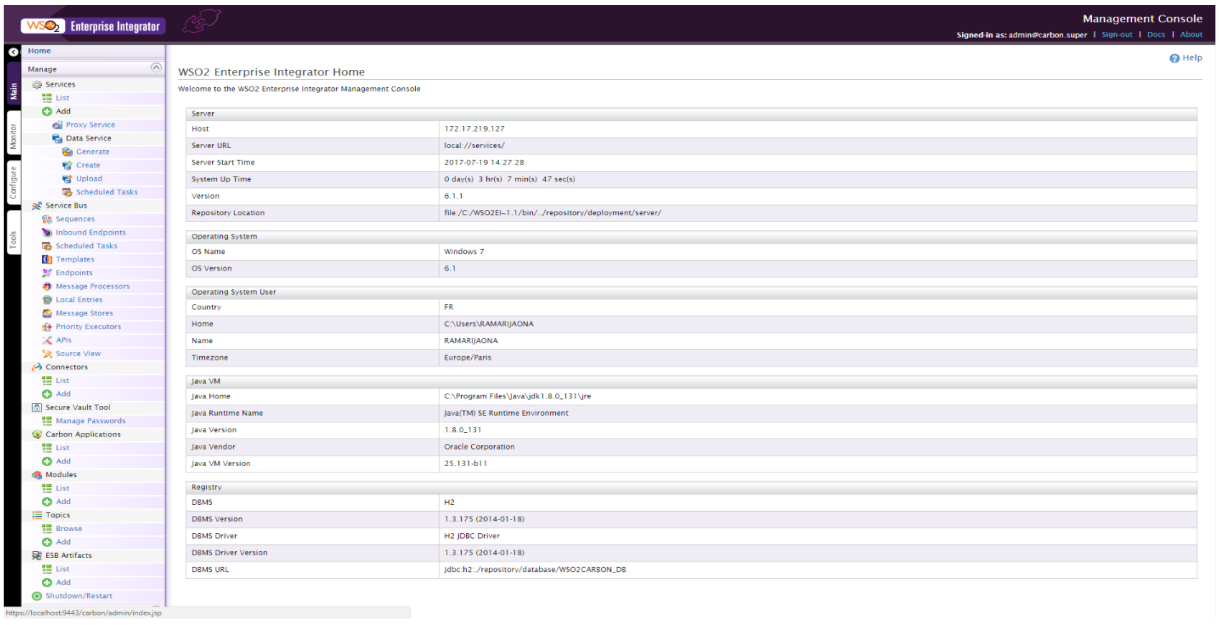
2. Installation de WSO2 Enterprise Integrator

L'installation de Enterprise Integrator nécessite JDK 1.8.*, et n'oubliez pas de définir la variable d'environnement JAVA_HOME.
Pour télécharger la version "on premise" qui est utilisée dans le tutoriel ci après, suivre ce lien : http://wso2.com/integration/, cliquer sur Download. Vous devrez ensuite renseigner quelques informations, puis télécharger le fichier .zip.

Dézipper le fichier dans un dossier de votre choix. Ce dossier sera désigné par $EI_HOME dans le reste des instructions de ce Wiki.

2.1. Lancement d' Enterprise Integrator

Ouvrez une invite de commande et déplacez vous dans $EI_HOME\bin . Exécutez ensuite integrator.bat .
Lorsque vous voyez le message INFO - StartupServiceFinalizerComponent WSO2 Carbon started in X sec , vous pouvez accéder à la Management Console d'Enterprise Integrator à l'adresse https://localhost:9443/carbon/ .
Vous serez en présence d'une page de login, les identifiants sont admin et admin, pour le nom d'utilisateur et le mot de passe.

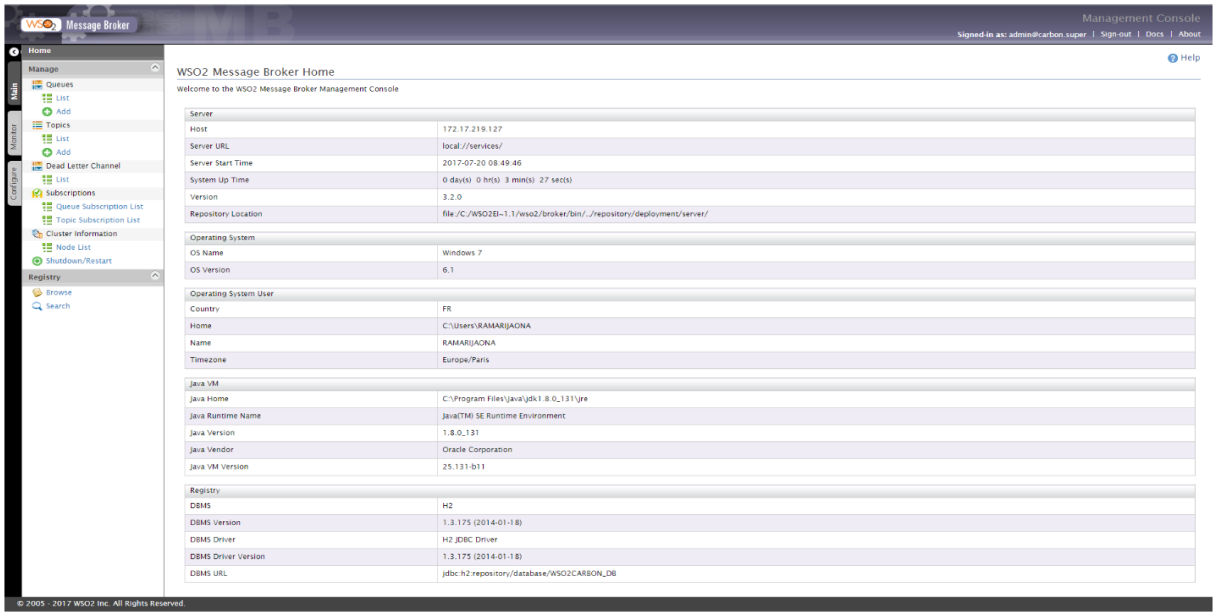


2.2. Lancement du Message Broker

Le Message Broker est la structure par laquelle passent les messages. Notamment, c'est ici que se trouvent les différentes queues où sont stockés les messages en attente d'être traités. Il faut activer ce profil **avant** de lancer Enterprise Integrator.

Avant de lancer le profil dans l'invite de commandes, vous devez effectuer quelques modifications.
Tout d'abord, Copiez-Collez depuis $EI_HOME\wso2\broker\client-lib les fichers : andes-client-3.2.19, geronimo-jms_1.1_spec-1.1.0.wso2v1 et org.wso2.securevault-1.0.0-wso2v2 vers $EI_HOME\lib.

Ouvrez une invite de commandes. Déplacez vous dans $EI_HOME\wso2\broker\bin et exécutez wso2server.bat. Lorsque le lancement est terminé, vous pouvez accéder à l'espace de gestion de WSO2 MB à l'adresse
https://localhost:9446/, les identifiants sont admin et admin pour le nom d'utilisateur et le mot de passe.



2.3. Analytics

Avant de lancer ce profil pour la première fois depuis l'invite de commandes, il faut effectuer quelques ajustements.
Dans les fichier $EI_HOME\conf\synapse.properties, mettez les paramètre suivants à true:
mediation.flow.statistics.enable, mediation.flow.statistics.tracer.collect.payloads, mediation.flow.statistics.tracer.collect.properties et mediation.flow.statistics.collect.all(lignes 63, 64, 65 et 70)

Le profil WSO2 EI Analytics permet de voir les statistiques sur les différents éléments présents dans votre profil Enterprise Integrator. Tout comme pour le message Broker, il convient de l'activer avant Enterprise Intergrator. Pour ce faire, il faut d'abord ouvrir une invite de commande, se déplacer dans $EI_HOME\wso2\analytics\bin et exécuter wso2server.bat . Lorsque le profil est lancé, vous pouvez accéder à la console de management à l'adresse https://localhost:9444/portal . Les identifiants sont admin et admin.

Pour voir les statistiques, cliquez sur l'option View. Vous verrez ensuite une fenêtre de ce type:

Pour voir les statistiques, cliquez sur l'option View. Vous verrez ensuite une fenêtre de ce type:

De haut en bas et de droite à gauche : Nombre de messages envoyés dans votre système, avec ensuite les pourcentages de réussite et d'échecs globaux, ensuite le nombre de traitements par seconde en fonction du temps et le nombre de succès et échecs en fonction du temps. Les différents "top five" sont dans l'ordre, les services proxy les plus utilisés, puis les API, puis les endpoints et inbound endpoints, et enfin les séquences les plus utilisées.

3. Structure de la maquette v1

1. L'API ANewAPI d'addresse http://localost:8280/anewappi/data_store (URI : http://{host}:{port}/{apiname}/{apiresource}) à laquelle on envoie via HTTP POST le fichier JSON que l'on souhaite stocker dans la base de données. (grâce à par exemple la commande curl -v POST http://localhost:8280/anewapi/data_store -d @examplerequest.json -h "Content-Type:application/json"). Cette ressource transmet le message au service proxy via l'"Endpoint Addresse" ANewAddressEP (il est important que l'endpoint soit un endpoint adresse et non un endpoint http même si l'adresse référencée par l'endpoint est http).
2. Service Proxy ANewProxyService
   Ce service proxy est chargé de placer le message dans la Queue ANewMessageStore dans le WSO2 Message Broker. Ce message est stocké dans la queue jusqu'à ce qu'un Message Processor le consomme.
3. Message Processor ANewMessageProcessor
   Cet élément va consommer les messages de ANewMessageStore et lui appliquer une suite de médiateurs contenus dans la séquence ANewSequence. Ce processor est du type Message Sampling Processor.

3. Message Processor ANewMessageProcessor
   Cet élément va consommer les messages de ANewMessageStore et lui appliquer une suite de médiateurs contenus dans la séquence ANewSequence. Ce processor est du type Message Sampling Processor.
4. Séquence ANewSequence
   Cette séquence va extraire du fichier JSON les informations importantes (ici le Tracking ID et le Machine ID) pour les transmettre au service de données via le protocole SOAP. Elle va d'abord extraire les propriétés trackingid et machineid, puis les placer dans un message SOAP grâce au médiateur PayloadFactory et enfin les envoyer au service de données via **l'Endpoint WSDL** de MongoMaquette.
5. Service de données MongoMaquette
   Ce web service permet de modifier la base de données Mongo. Elle a deux queries définies: verify_data qui permet de voir le contenu de la base et add_parcel qui permet d'ajouter des informations dans la base. Ces queries ont des opérations associées add_parcel_op et verify_data_op qui peuvent être invoquées grâce à des requêtes SOAP. L'opération qui nous intéresse ici est add_parcel_op. Le message composé dans le médiateur PayloadFactory de la séquence permet d'appeler add_parcel_op avec les paramètres attendus. Ainsi, lorsque ce message est reçu par le service, les informatios sont ajoutées à la base de données.

4. La maquette en détail

4.1. Le service data MongoMaquette

Pour construire un data service, il faut être connecté sur la console de gestion Enterprise Integrator. Lancez integrator.bat depuis $EI_HOME\bin puis connectez vous sur http://localhost:9443/ avec les identifiants admin pour le nom d'utilisateur et le mot de passe.

Dans la colonne de gauche cliquez sur Services > Data services > Create.

Donnez un nom unique à votre service (dans l'exemple : MongoMaquette) et gardez les transport settings à http et https. Puis cliquez sur Next pour ajouter une base de données.

On suppose ici que vous avez déjà une base de donnée MongoDB en place. Cliquez sur Add New Datasource. Donnez un nom unique à cette source (ici mongomaq) et déléctionnez MongoDB dans le champ Datasource Type. Entrez ensuite l'addresse et le port du serveur où se trouve votre base de données, et le nom de la base que vous utilisez (ici localhost:27017 et mydb). Enfin sélectionnez les options suivantes pour les autres champs:

Write concern : NORMAL
Read preference : PRIMARY
Authentication Method : NONE

Les autres champs sont laissées vides. Cliqyez sur Save puis Next pour ajouter des requêtes.
Il y a besoin de deux opérations sur cette base de données pour ce projet : ajouter des éléments qui contiennent un MachineId et un TrackingId, et vérifier les TrackingId des lettres traitées par la machine 3.

Cliquez sur Add New Query, commencez par la requête qui servira à ajouter les éléments. Donnez un nom unique à la requête (ici add_parcel) et sélectionnez mongomaq comme datasource.

L'expression de cette requête est :
maquette.insert("{TrackingId:#, MachineId:#}") (l'exemple utilise une collection MongoDB appelée maquette)

Il faut ensuite ajouter des paramètres d'Input Mapping que la requête mettra à la place des #. La première est TrackingId. Son mapping name est donc TrackingId. C'est un paramètre de type SCALAR, STRING et IN. Les types seront les mêmes pour le paramètre MachineId. Cliquez sur Save lorsque vous avez terminé de définir chaque paramètre. Cliquez ensuite sur Main Configuration puis sur Save pour ajouter une autre requête.

Le seconde requête pose encore des problèmes notamment quant à l'utilisation des différents champs d'output mapping

Il faut maintenant faire en sorte qu'on puisse effectuer ces requêtes en appelant le service MongoMaquette depuis un autre service. Il y a deux choix possibles au premier abord: soit utiliser le service comme une ressource REST soit comme un récepteur de messages SOAP. Dans ce projet, l'approche SOAP sera utilisée. *L'ajout d'élément ne fonctionne pas en REST et la vérification les éléments déjà présents retourne les données dans un format illisible*

Pour cela, il faut créer des opérations correspondant aux différentes requêtes qui ont été définies. Cliquez sur Next à partir de l'écran d'affichage des différentes requêtes pour ajouter des opérations, puis sur Add New Operation. Commencez par l'opération qui correspond à la requête add_parcel. Tout d'abord, donnez un nom unique à votre opération (ici add_parcel_op) et sélectionnez add_parcel pour le champ Query ID. Cliquez sur Save.
La configuration de l'autre opération se fera de manière similaire

Cliquez sur Finish pour enregistrer les modifications apportées au service. En cliquant sur Services > List à gauche de l'écran de la console de gestion, vous pourrez voir que votre Service de données MongoMaquette a bien été déployé.

4.2. La solution WSO2

La solution ANewSolution comprend le projet ANewSolutionProject et l'application composite ANewSolutionCompositeApplication. Pour créer cette solution avec WSO2 Tooling, il faut aller dans Developper Studio (dans la barre d'outils) > Open Dashboard > ESB Solution Project. Ensuite, il suffit de rentrer le nom du projet et garder les options par défaut. La composite application est utile pour déployer le projet grâce à la Management Console de Enterprise Integrator, les Registry resources permettent de stocker les différents Endpoints, Sequences et autres services que nous créeront par la suite. (Le Connector Exporter pourra être utile si l'on décide d'augmenter les scénarios d'utilisation de ce projet.)

4.3. API ANewAPI

C'est cette partie que le client contacte pour déclencher la chaîne d'opérations à effectuer sur le fichier JSON.
A partir de WSO2 Tooling, on peut la créer en faisant clic droit sur le dossier du projet dans l'explorateur de projet, puis New > REST API.

Ici NewAPI a été créée mettant comme nom "ANewAPI" et comme contexte "anewapi".

Une API vide est donc créée.

Avec la palette, on peut ajouter une ressource API, c'est à dire une première chaîne d'actions à effectuer. Cette ressource sera de type URI TEMPLATE et son Template sera "/data_store", le protocole est http, https et la sule méthode authorisée sera POST. Vous pouvez changer toutes ces propriétés en faisant clic droit sur l'icone de la ressource puis Properties View.

Après avoir créé cette ressource, on peut ajouter un élément Log pour conformer l'entrée du message dans l'API. Pour ce faire cliquez d'abord dans la Palette sur l'onglet Médiators et cliquez-déplacez un médiateur Log dans la ressource data_store.

Après avoir enregistré le log, le message sera envoyé au service suivant pour commencer le traitement. Déplacez un médiateur Send depuis la palette et placez le après le médiateur Log.

Pour l'instant, comme aucun autre service n'a été créé, le médiateur Send sera laissé vide.

4.4. Service Proxy ANewProxy et endpoint correspondant

Maintenant il faut créer le service proxy qui va effectivement placer les message dans une message queue.

Pour ce faire, il faut tout d'abord faire clic droit sur le dossier du projet dans le project explorer et séléctionner New > Proxy service. Ensuite, donner un nom au service. L'adresse de ce service sera donc http://{host}:{port}/services/ANewProxyService.

Pour commencer, cliquez déplacez un médiateur Log dans le service. Ensuite ajoutez un médiateur Store (toujours dans l'onglet Médiators) et finissez par un médiateur Drop. En effet, lorsque le message est placé dans la queue, le proxy a terminé son travail et n'a plus à s'occuper de ce message.

Vous pouvez maintenant passer à la construction de l'endpoint correspondant à ce service proxy. Faites clic droit dans le Project Explorer sur le dossier du projet et choisissez New > Endpoint. Choisissez de créer un Adress Endpoint. Cela signifie que lorsque ce Endpoint sera utilisé, le message sera transmis à l'adresse mentionnée. L'URI à donner pour la création de cet endpoint est http://localhost:8280/services/ANewProxyService. Donnez également un nom unique à l'endpoint.

Après la création de cet endpoint vous pouvez revenir à l'API ANewAPI. Dans la palette, ouvrez l'onglet "Defined Endpoints", vous y trouverez l'endpoint que vous venez de définir. Cliquez-déplacez cet élément dans le médiateur Send déjà présents dans l'API.

Maintenant, quand le message est envoyé à l'API, il est transmis au service proxy qui lui même le place dans la message queue.

4.5. Message Store

Le message doit être stocké dans un message store. Pour créer ce store, il suffit de faire clic droit sur le dossier du projet dans l'explorer et choisir New > Message Store. Choisissez comme type de store WSO2 MB Message Store. En effet, ce type de message store utilise le message broker fourni avec Enterprise Integrator.

Ensuite, entrez les valeurs suivantes
Initial Context Factory : org.wso2.andes.jndi.PropertiesFileInitialContextFactory
Queue Connection Factory : amqp://admin:admin@clientID/carbon?brokerlist='tcp://localhost:5675'

Enfin pour JNDI Queue Name, mettez le nom que vous avez donné à votre store.

Il faut également modifier le fichier $WSO2_HOME/conf/jndi.properties pour effectivement ajouter le store dans le WSO2 Message Broker.

Ajoutez la ligne: queue.{nom_de_votre_sotre} = {nom_de_votre_store} dans jndi.properties .

Revenez à l'écran de votre service proxy et faites clic droit sur le médiateur Store présent dans ce service. Sélectionnez Properties View et entrez le nom de votre messages store dans les champs Available Message Store et Message Store.

4.6. Message Processor

Les messages peuvent maintenant être stockés dans le message store, il faut maintenant un message processor qui extrait les messages et les traite.

Faites clic droit sur le dossier du projet dans le project explorer et sélectionnez New > Message Processor. Ce projet utilise un Message Sampling Processor, c'est à dire un processor qui va prendre un certain nombre de messages à intervalles de temps régulier et qui va leur appliquer un Séquence.

La propriété Smapling Interval est le laps de temps que le procssor attend entre deux prélèvement de messages et Processor Concurrency est le nombre de messages à prélever à chaque fois. Choisissez des valeurs adaptées à l'utilisation que vous souhaitez faire du projet. (En général, la valeur maximale de Concurrency pour 1 message processor est de 100)

Après avoir prélevé les messages, le processor les envoie à une séquence qui va les traiter.

4.7. Sequence ANewSequence

Une séquence est une suite d'opérations qui peut être appelée par d'autres éléments du projet mais pas depuis l'extérieur, contrairement à un service proxy ou à une api.

Vous allez devoir construire la séquence appelée par le message Processor. Cette séquence va construire le nouveau message à transmettre au service de base de données Mongo et l'envoyer à ce service.

Faites un clic droit sur le dossier du projet et Sélectionnez New > Sequence.

Commencez par ajouter un médiateur Log dans la séquence.

Ensuite, il faut extraire les deux propriétés importantes du message, MachineId et TrackingId.
Déplacez deux médiateurs Property dans la séquence. Pour modifier les propriétés de ces médiateurs, faites un clic droit dessus et sélectionnez Properties View.

Ensuite remplissez les différents champs comme ceci:

Property name : New Property...
New Property name : machineid
Property action : set
Value type : EXPRESSION

Value type : EXPRESSION
Property Data type: STRING
value expression : json-eval($.Parcel.PSM.MachineId) (càd : la valeur du médiateur est la valeur du champ Parcel.PSM.MachineId du fichier JSON contenu dans le message)
Description :set MachineId

Faire de même pour l'autre médiateur Properties en remplaçant MachineId par TrackingId.

Le message est transmis au service par le protocole SOAP au format WSDL. Il faut donc utiliser le médiateur PayloadFactory pour transformer le message depuis JSON vers du WSDL.

Cliquez déplacez le médiateur PayloadFactory depuis la palette et placez le après les deux médiateurs Property. En effet la fabrique de message va utiliser ces deux propriétés.

Configurez le médiateur comme suit:
Payload Format : inline
Media Type : xml
Payload :<p:add_parcel_op xmlns:p="http://ws.wso2.org/dataservice">
<xs:TrackingId xmlns:xs="http://ws.wso2.org/dataservice">$1</xs:TrackingId>
<xs:MachineId xmlns:xs="http://ws.wso2.org/dataservice">$2</xs:MachineId>
</p:add_parcel_op>

Il faut ensuite préciser à quoi correspondent ces variables $1 et $2. Pour ce faire, cliquez sur le champ Args de la Property view du médiateur PayloadFactory et ajoutez deux variables de type expression, d'évaluateur xml. Pour la valeur de l'expression à évaluer pour la première variable, entrez : $ctx:trackingId et pour la deuxième : $ctx:machineId .

Le message résultant contient donc deux paramètres, TrackingId et MachineId tous deux tirés du message JSON original.

Après le médiateur PayloadFactory, ajoutez un médiateur Call (on le laisse vide pour le moment). Le médiateur Call, tout comme le médiateur Send, permet d'envoyer le message à un autre service mais contrairement à Send, il permet de placer d'autres médiateurs ensuite.

Placez un médiateur Log pour s'assurer que le message a bien été envoyé au service demandé et enfin un médiateur Drop car cette séquence n'a plus besoin de traiter ce message après l'avoir envoyé au service.

4.8. L'endpoint vers le service MongoMaquette

La séquence que vous venez de construire va envoyer le message au service data MongoMaquette créé au début du tutoriel. Pour ce faire, il faut créer un endpoint de ce service que la séquence pourra appeler.

Faites clic droit sur le dossier du projet dans le project explorer et sélectionnez New > Endpoint. Comme le protocole utilisé sera SOAP, il convient de créer un WSDL Endpoint. Choisissez cette option dans la fenêtre de création d'endpoint. Donnez un nom unique à votre endpoint (ici MongoMaqWSDLEP), puis sélectionnez comme option de format LEAVE_AS_IS et activez la trace et les statistiques.

Pour savoir les autres paramètres, visitez la page de gestion de Enterprise integrator (https://localhost:9443/carbon/) et cliquez sur Services > List sur le menu de gauche.

Pour connaître le paramètre WSDL URI, sélectionnez WSDL 1.1 sur la ligne de votre service MongoMaquette. La cible de ce lien est l'adresse à entrer dans le champ WSDL URI de la création de l'endpoint.
Pour le paramètre Service, entrez le nom de votre service (ici MongoMaquette)

Sur la ligne de votre service MongoMaquette, sélectionnez Try this service. Une fois sur la page d'essai de votre service, sélectionnez l'opération add_parcel_op (celle qui permet d'ajouter des informations dans la base de données. La première ligne de la page ("Using Endpoint:") vous donne la valeur à rentrer dans le champ Port de la fenêtre de création.

Quand vous avez terminé, vous pouvez revenir à la Séquence (ANewSequence) et sélectionner MongoMaqWSDLEP dans l'onglet Defined Endpoints de la Palette. Placez le dans le médiateur Call de la séquence.

4.9. L'archive .car

Maintenant que tous les éléments du projet sont construits, il faut les regrouper dans une Composite Application Archive (fichier.car) pour pouvoir ensuite la déployer grâce à la console de gestion d'Enterprise Integrator.

Dans l'explorateur de projet, sélectionnez {le_nom_de_votre_projet}CompositeApplication > pom.xml. Vérifiez que la case portant l'étiquette {nom_de_votre_projet} est cochée. Pour vous assurer que tous les éléments que vous avez créés seront bien empaquetés dans l'archive, cliquez sur la flèche à côté de la case pour dérouler la liste des éléments et constatez que tous vos endpoints, séquences, APIs et services proxy sont cochés.
Après avoir modifié si nécessaire le numéro de version, vous pouvez ormer l'archive en cliquant sur le bouton Create Archive en haut à droite de l'écran. Choisissez ensuite un dossier où former l'archive.

Pour déployer l'archive avec tous les éléments qu'elle contient, ouvrez la console de gestion de Enterprise Integrator et sélectionnez dans le menu de gauche Carbon Applications > Add. Cliquez sur Choose a File et naviguez jusqu'à l'endroit où vous avez formé l'archive. Cliquez ensuite sur Upload.
Attendez quelques secondes puis cliquez dans le menu de gauche sur Carbon Application > List. Vous verrez que votre projet a été déployé. Vous pouvez cliquer sur le nom de votre Composite Application pour voir la liste des éléments qui composent votre projet.

5. Premiers tests

5.1. Envoi d'un message via cURL

Si vous ne disposez pas déjà de cURL, vous pouvez vous le procurer sur https://curl.haxx.se/ .

Voici un exemple de message JSON à envoyer au système:

Pour tester le système, activez le profil Message Broker, puis le profil Analytics et enfin Enterprise Integrator. Puis envoyez le message JSON grâce à la commande suivante:

curl -v http://localhost:8280/anewapi/data_store --data @examplerequest1.json --header "Content-Type:application/json"

5. Premiers tests

5.1. Envoi d'un message via cURL

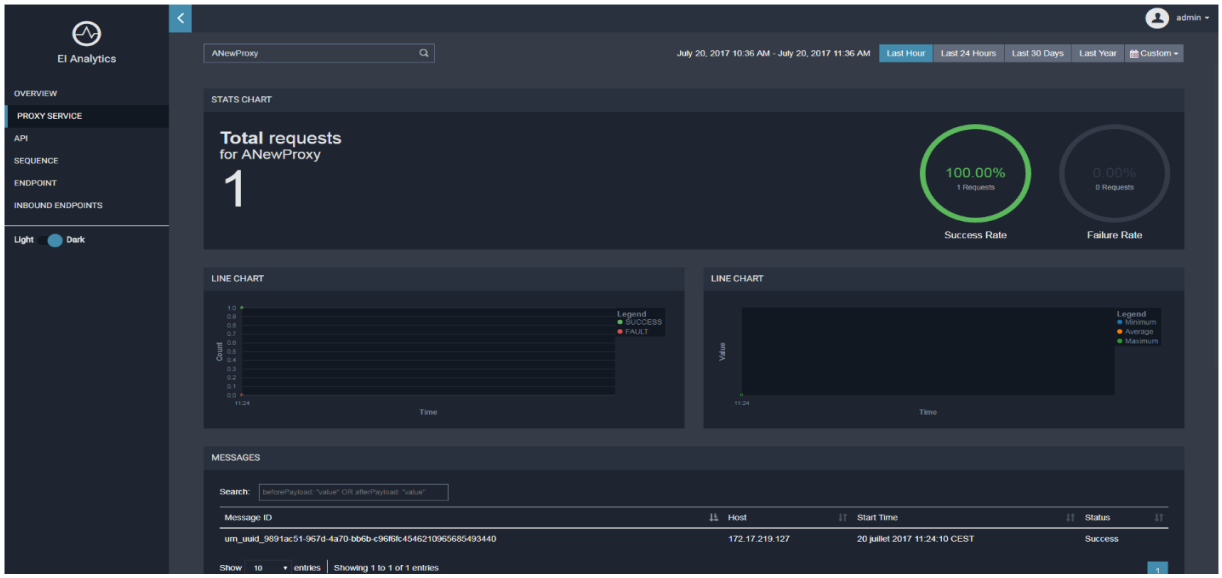
Si vous ne disposez pas déjà de cURL, vous pouvez vous le procurer sur https://curl.haxx.se/ .

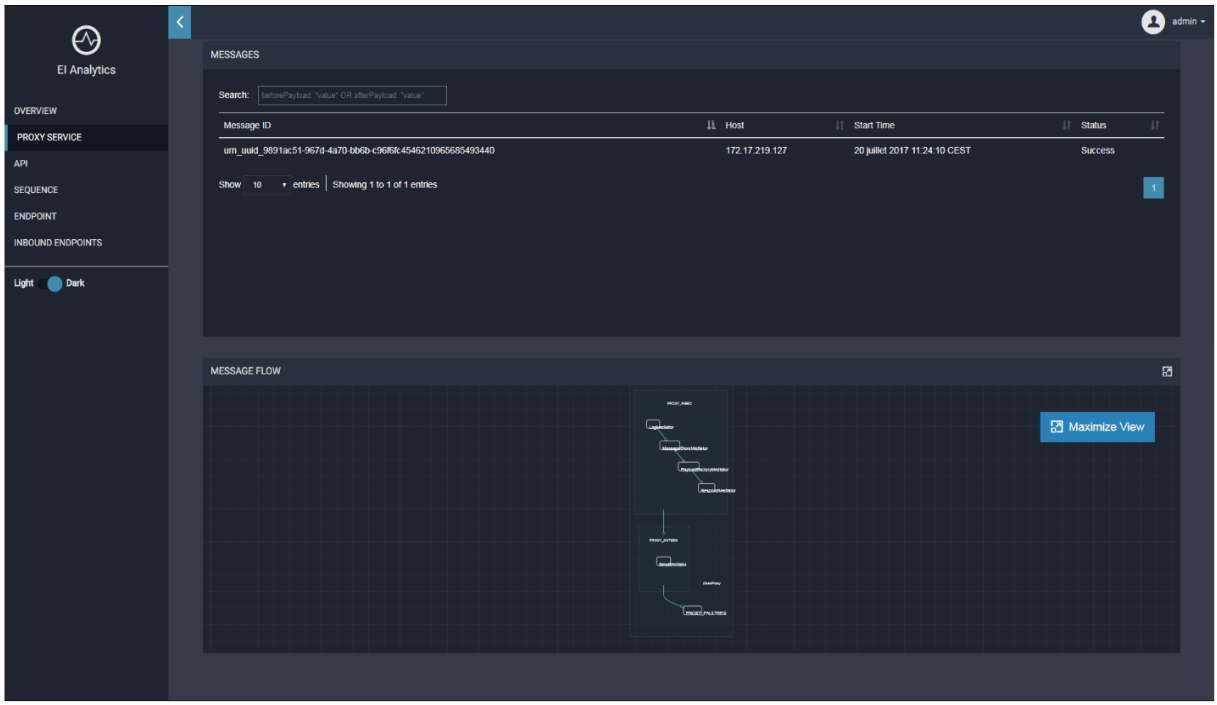
Voici un exemple de message JSON à envoyer au système:

Pour tester le système, activez le profil Message Broker, puis le profil Analytics et enfin Enterprise Integrator. Puis envoyez le message JSON grâce à la commande suivante:

curl -v http://localhost:8280/anewapi/data_store --data @examplerequest1.json --header "Content-Type:application/json"

Attendez ensuite quelques instants puis connectez vous sur la console de EI Analytics. Vous pourrez voir alors qu'un message est passé dans le système. Depuis l'écran Overview, vous pouvez cliquer sur n'importe quel graphique en barres pour obtenir des informations supplémentaires sur un certain élément. Par exemple, si vous cliquez sur la barre ANewProxySrevice, vous verrez qu'un message est passé par ce service.





Si vous cliquez sur le message ID dans la section Messages, vous pourrez voir des informations complémentaires, comme par exemple le payload (contenu) du message, la valeur des différentes propriétés ou le nom du dernier endpoint traversé.

Visuals from the SoMoS II application

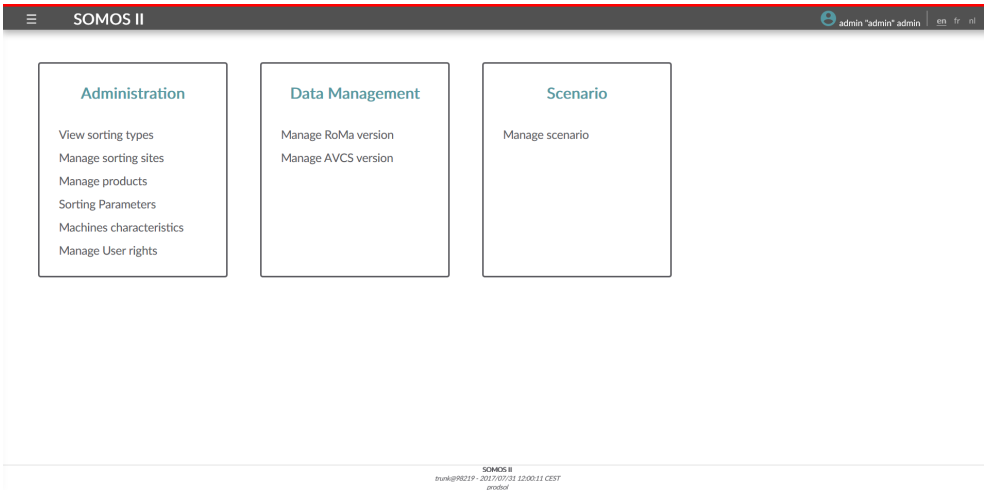


Figure 6: SoMoS II Homepage

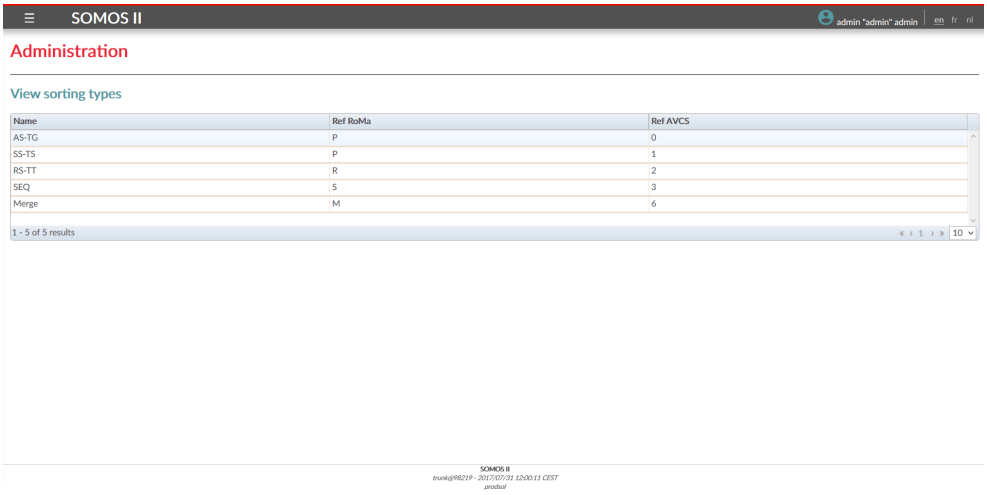


Figure 7: A PRM page

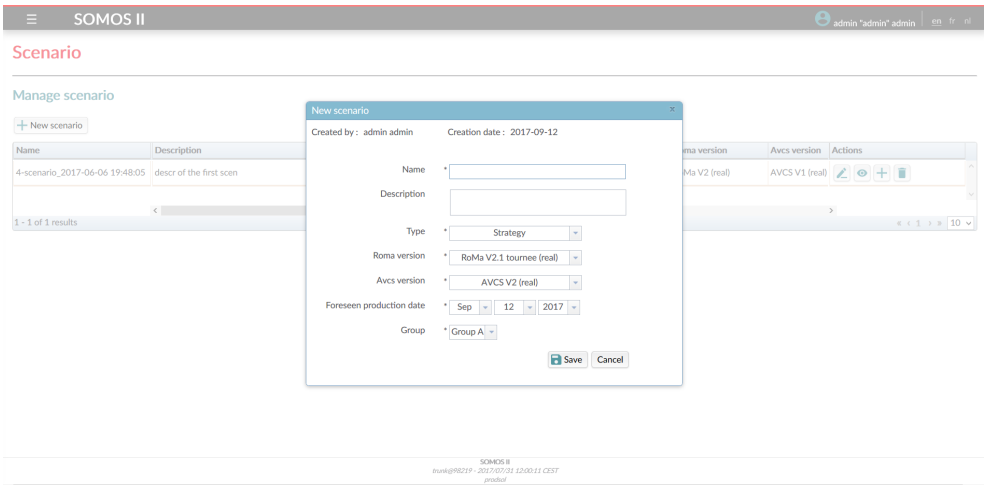


Figure 8: Creating a Scenario

Visuals from WSO2 Tooling

WSO2 Tooling is the environment that allows a developer to configure WSO2 EI, to create APIs, proxy services and endpoints (refer to the Wiki to learn more about what these are). Here are some screenshots of the GUI. All of the strucutres shown come from the ”training application” that i built to learn about WSO2.

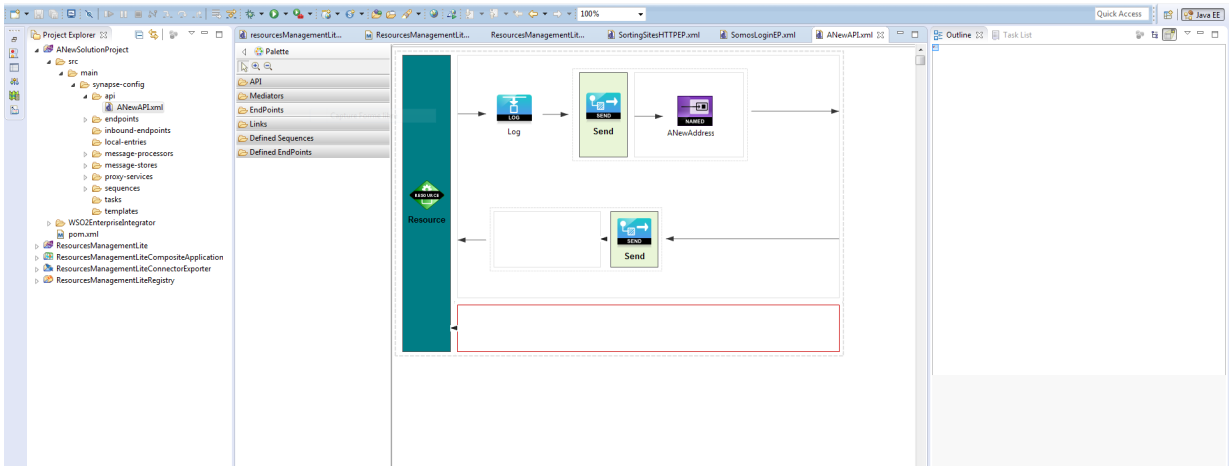


Figure 9: The layout of the GUI (Eclipse based)

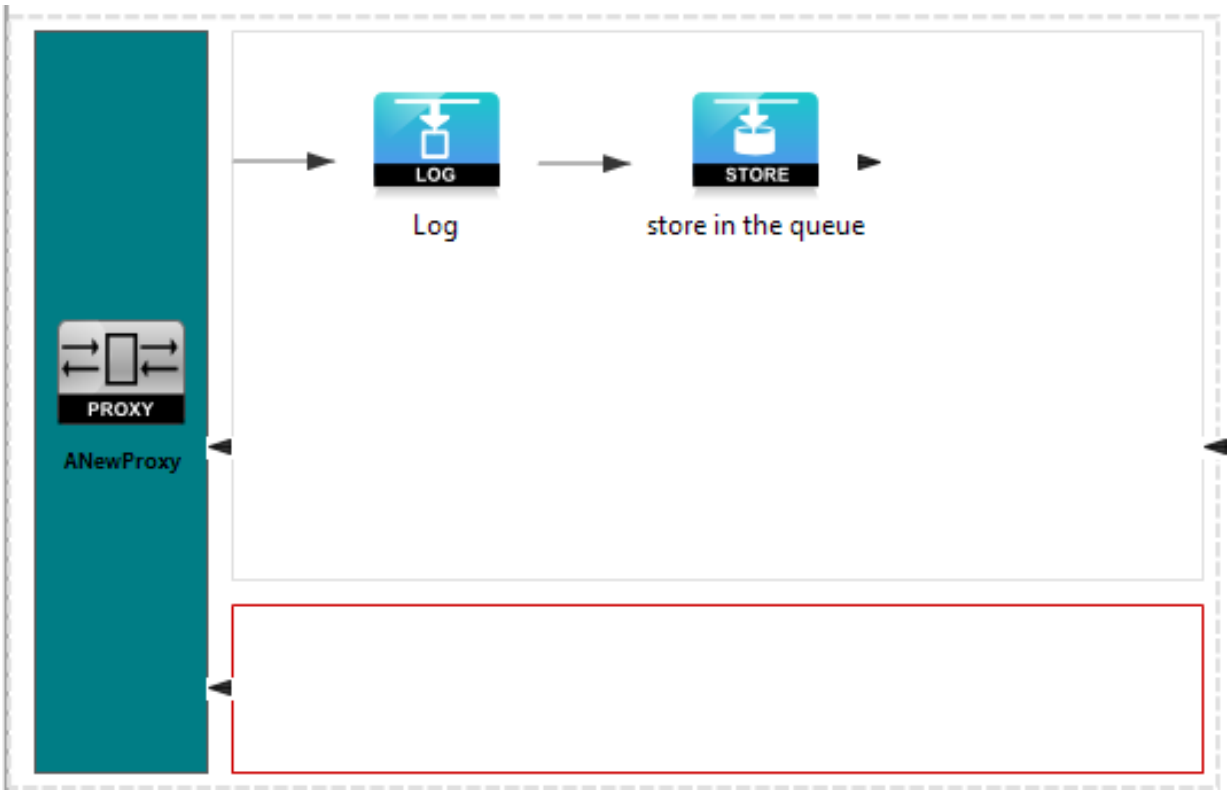


Figure 10: A service that stores a message in a queue

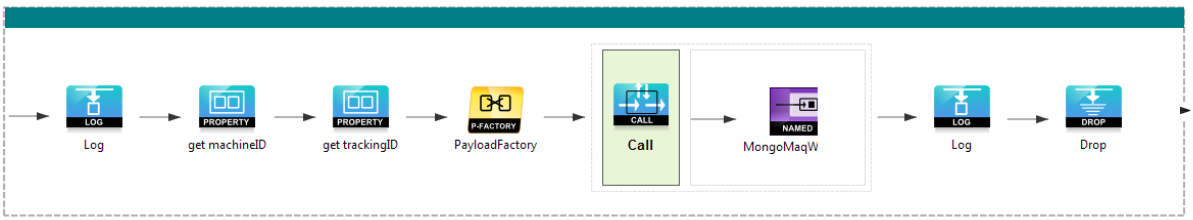


Figure 11: A sequence