DOSSIER DE SECURISATION DES WEB SERVICES

ANNEXE 4 : ETUDE DE SYNAPSE

ABREVIATIONS

Abréviation	Signification
SOA	Service Oriented Architecture
HTTP	Hyper Text Transfer Protocol
SOAP	Simple Object Access Protocol
ESB	Enterprise Service Bus
WSDL	Web Services Description Language
JCA	Java Connector Architecture
XML	Extensible Markup Language
SLA	Service Level Agreement
BPEL	Business Process Execution Language
EDI	Electronic Data Interchange
SDO	Service Data Objects
WS-RM	Web Services Reliable Messaging Protocol
SCA1	Software Communications Architecture
JBI3	Java Business Integration
JMS	Java Message Service
QOS	Quality of Service

REFERENCES

Abréviation	Signification
[DOSSIERSECWS]	Dossier_Securisation_Web_Services_v1r0.pdf
[ANASEC]	Annexe1_Analyse_Sécurité_Préalable_v1r0.pdf
[FILTRAGE]	Annexe2_Etude_ Securistation_WS_FiltrageIP_v1r0.pdf
[SSL]	Annexe3_Etude_ Securistation_WS_TunnelSSL_v1r0.pdf
[SYNAPSE]	Annexe4_Etude_ Securistation_WS_Synapse_v1r0.pdf
[SPRING]	Annexe5_Etude_ Securistation_WS_SPRING-Security_v1r0.pdf
[PROTOTYPE]	Annexe6_Etude_ Securistation_WS_Prototype_v1r0.pdf

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

TABLE DES LISTINGS

1.AVANT PROPOS

Ce document constitue une annexe du « Dossier de sécurisation des Web Services » publié par l'AMUE. Il a pour objectif de présenter la mise en œuvre de Synapse dans un objectif de sécurisation des Web Services. Par conséquent, afin d'appréhender correctement le contexte et le contenu de ce document, il est conseillé de lire au préalable le dossier de sécurisation des Web Services [DOSSIERSECWS].

2. INTRODUCTION AUX ESB (ENTERPRISE SERVICE BUS)

2.1. DESCRIPTION GENERALE

D'un point de vue technique, le rôle d'un ESB est la connexion et la médiation entre les services et applications du Système d'Informations. C'est un intermédiaire qui permet à des applications hétérogènes de communiquer au travers de protocoles standards. A ce titre, ses principales responsabilités sont les suivantes :

- La réconciliation des mondes hétérogènes, à l'aide de standards d'interopérabilité ou de connecteurs spécialisés – c'est le rôle classique d'un middleware d'intégration.
- Découpler consommateurs et fournisseurs de services : le consommateur ne connaît que l'ESB, mais ne connaît ni les formats, ni les protocoles d'échanges spécifiques utilisés par le fournisseur du service.
- Agréger des services de niveau N afin de construire des services de niveau N+1. Si l'agrégation est complexe, ou nécessite des structures de contrôle du flux d'exécution, un moteur d'orchestration reposant par exemple sur le langage BPEL (Business Process Execution Language), est mis à contribution.
- Tracer les messages qui transitent. Devenant une zone de passage incontournable, l'ESB joue un rôle fondamental dans la traçabilité et le monitoring des traitements.

2.2. FONCTIONNALITES

2.2.1. ADAPTATION AUX ENVIRONNEMENTS HETEROGENES

L'ESB apporte une couche d'abstraction vis à vis des technologies utilisées dans le SI et doit en conséquence s'adapter aux protocoles et aux formats d'échanges des applications qui le composent. Par exemple, un ESB :

- Ne dépend pas d'un système d'exploitation ou d'un langage ;
- Supporte autant de standards que possible, dont les Web Services, XML, JCA (et dans l'avenir SCA1 / SDO2 et / ou JBI3) ;
- Permet d'exposer les services de manière uniforme quelque soit la technologie sous-jacente ;
- Utilise XML (ou SDO) comme langage standard de représentation et de traitement des données ;
- Offre un panel ouvert de connecteurs spécialisés vers les différentes briques du SI (MainFrames, application propriétaire, progiciel, etc.).

2.2.2. MEDIATION ET ROUTAGE

La médiation est la principale valeur ajoutée d'un ESB. Dans ce domaine, il doit donc proposer une sémantique riche, susceptible de fiabiliser et de simplifier les échanges, tout en offrant une grande souplesse de mise en œuvre. Voici les principales fonctionnalités que l'on peut attendre d'un ESB en matière de médiation et de routage :

- Support de différentes sémantiques d'échange de messages (synchrone « requête-réponse », asynchrone point-à-point, asynchrone selon le modèle « publication-souscription ») ;
- Gestion de règles de routage sur les messages ;
- Mécanismes de gestion de la priorité des messages ;
- Services de transformation et de conversion de messages (ex : XML <-> EDI, etc.) ;
- Manipulation des messages (enrichissement, transformation, combinaison, découpage) ;
- Validation des données entrantes ou sortantes ;
- Gestion de versions de services de façon transparente (systèmes évoluant à des rythmes différents).

2.2.3. MANAGEMENT, MONITORING, CONTRAT DE SERVICE

Étant un point central, l'ESB doit permettre le suivi et la fiabilisation des échanges qui transitent en son sein grâce notamment à :

- La garantie de livraison des messages tout en conservant les messages non consommés (application indisponible, échec d'une transaction ...) ;
- Des fonctionnalités permettant de suivre les traitements effectués et les messages reçus ;
- Une gestion de la sécurisation des services ;
- Contrôle des SLA (Service Level Agreement) et aptitude à modifier le comportement du bus (priorités ...) pour assurer ces SLA.

3.INTRODUCTION A WSO2 ESB (SYNAPSE)

De plus en plus d'entreprises commencent à intégrer des systèmes métiers. La preuve de cette approche est l'utilisation des Architectures Orientées Service (SOA : Service Oriented Architecture). Après un bref aperçu des possibilités offertes par les ESB en général, nous nous attarderons ici sur un ESB en particulier : WSO2 ESB. Ce dernier a été retenu car il est une solution open-source mature et que sa communauté est active (Apache)

WSO2 Enterprise Service Bus (WSO2 ESB) offre une nouvelle approche légère de création d'une SOA, et y ajoute des outils de surveillance, de gestion et de virtualisation aux services existants. En fait, il facilite la construction d'un SOA simple, efficace et surtout pragmatique. Qu'en est-il réellement ?

3.1. DESCRIPTION GENERALE DE LA TECHNOLOGIE

WSO2 ESB, solution Open Source basée sur une Licence Apache 2.0 (Il ne s'agit en effet pas d'un logiciel propriétaire), repose sur la collaboration entre la société Sri-lankaise WSO2 et le projet Apache Synapse. Il a été conçu pour permettre d'effectuer un contrôle sur les systèmes d'intégration, du simple monitoring d'un service et des messages d'interaction en utilisant la console d'administration web aux fonctionnalités plus complexes comme le routage d'un contenu ou même la sécurité.

Il est construit autour des standards ouverts tels que http/s, XML, JMS (Java Message Service), JavaScript, Java, SOAP, WS-RM, WS-Security…

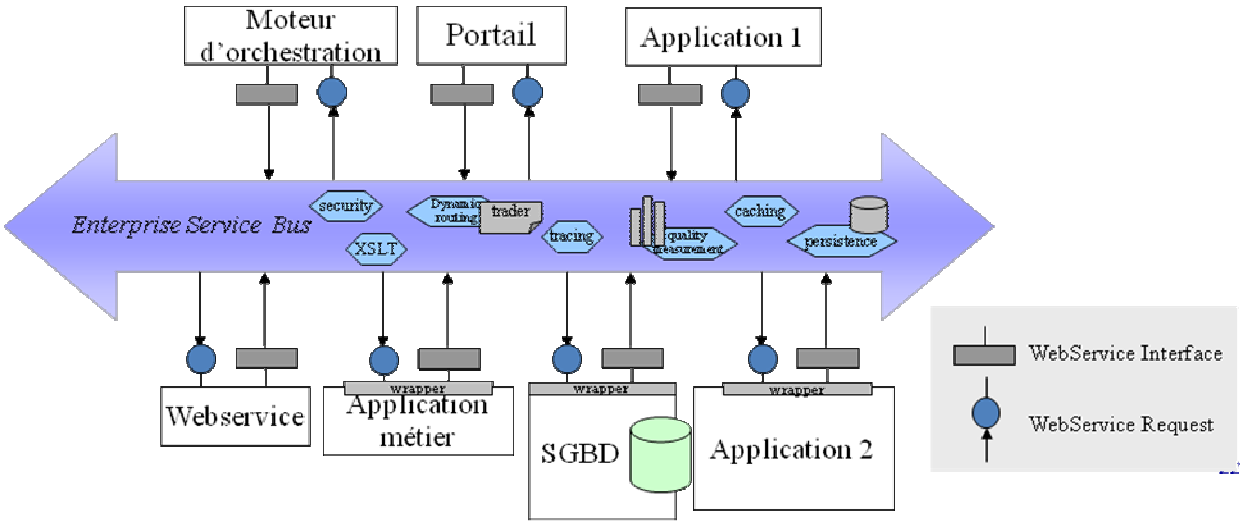

Figure 1 : Principe d'un ESB

3.1.1. LES FONCTIONNALITES

La solution WSO2 propose les fonctionnalités standards des ESB (non exhaustive)

- Répartition de charge
- Fail-over
- Proxy HTTP
- Routage
- Découverte de services
- Concentrateur de Web Services

3.1.1.1. Services Proxy

La façon la plus simple de commencer est d'utiliser l'assistant des services proxy qui permet à l'administrateur de créer des services virtuels grâce à un assistant graphique simple d'utilisation.

Le service proxy permet l'autorisation d'accès au service virtuel pour de multiples transports, mais il supporte également le WS-ReliableMessaging (WS-RM) et le WS-Security (WSS). WS-ReliableMessaging intègre dans les messages Soap des mécanismes d'envoi d'accusés de réception et de réémission en cas d'incident. Cette spécification permet donc de s'assurer qu'un message Soap transporté sur HTTP arrive à destination.

Ce modèle aboutit à la virtualisation de services, permettant aux clients d'être détachés de l'implémentation actuelle du service.

Les services proxy permettent aussi la commutation de transport (ex : de XML/JMS vers SOAP/HTTP vers XML/HTTP etc.), aussi bien que la répartition de charge (load-balancing) et le fail-over.

- Pendant le load-balancing, WSO2 ESB répartit la charge parmi les « end point » dits de load-balancing en fonction de l'algorithme de répartition de charges.
- Pour ce qui est du fail-over, il existe plusieurs sections de « end points » désignées comme groupes de fail-over. Lorsqu'un « end point » particulier échoue lors de l'accomplissement de son service, WSO2 ESB acheminera alors le message vers les autres « end point » du groupe.

La console d'administration est un moyen simple de contrôler et de tracer les services proxy.

3.1.1.2. Policy-based Mediation

De plus le WSO2 ESB peut agir comme un proxy HTTP. Cela permet aux clients d'être simplement redirigés pour envoyer des messages via l'ESB, où ils peuvent être manipulés en utilisant une approche à base de « policy » (règle à appliquer).

Ce modèle permet de vérifier un jeu de règles sur les messages, comme par exemple les fautes de frappe.

Les règles peuvent utiliser des techniques comme des expressions Xpath ou des expressions régulières (regex) pour appliquer des modifications aux messages particuliers.

3.1.1.3. Effective Partner Integration

ReliableMessaging, WS-SECURITY et HTTPS étant des briques par défaut de WSO2 ESB il est possible d'exposer des services internes sachant que WSO2 ESB gérera la sécurité et l'intégration fiable sans aucun rajout de code.

L'approche « services proxy » permet d'exposer des services internes de façon simple et rapide et permet aussi de gérer la sécurité, l'identification et le contrôle en un même endroit.

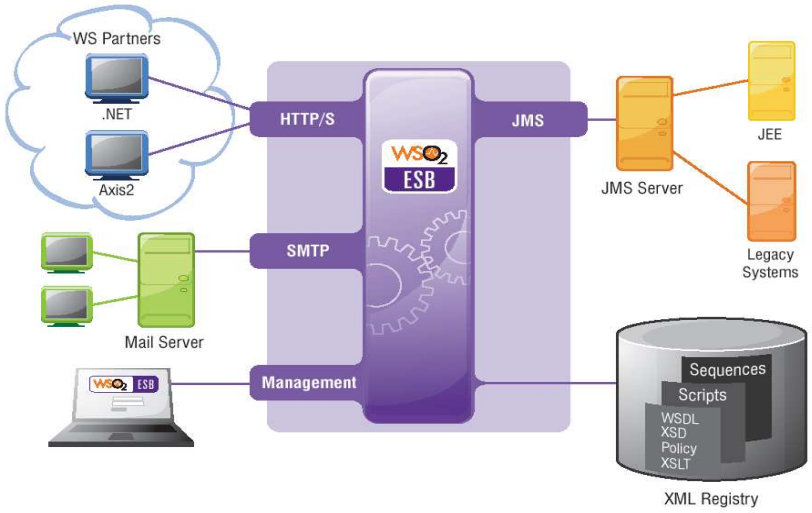

Figure 2 : Architecture d'un WSO2 ESB

3.1.1.4. Extensible Mediation Framework

Comme le WSO2 ESB est basé sur le framework open-source Synapse d'Apache, il a un modèle d'extensibilité simple et ouvert. Supportant Java, JavaScript, Ruby, Groovy et Spring, on peut même étendre le cœur des médiateurs avec du code XML personnalisé.

3.1.1.5. Built-in Registry

Le « Built-in Registry » permet aux ressources de configuration XML d'être cataloguées et gérées de manière centralisée.

L'enregistrement prévoit le stockage de définitions de séquences, de schémas, de scénarii, de transformations, de « policy », etc. et les rend disponibles pour exécuter des instances WSO2 ESB.

De plus, l'enregistrement supporte le rechargement dynamique de ressources, permettant aux administrateurs de gérer le SOA sans devoir éteindre l'ESB puis le redémarrer.

L'ESB supporte également l'accès aux registres distants via HTTP ou d'autres fournisseurs d'URL.

3.1.2. INSTALLATION

3.1.2.1. Pré requis techniques

Il est nécessaire de suivre les instructions suivantes avant d'utiliser la solution WSO2 ESB.

- OS compatibles : Linux, Solaris, MS Windows - XP/ Vista (Notons ici que tous les tests n'ont pas été effectués avec cette version). Comme WSO2 ESB est une application Java, il est généralement possible de l'exécuter sur d'autres plateformes disposant d'un JDK 1.5.x. Cependant, Linux/Solaris est recommandé pour un déploiement dans un environnement de production.
- JDK 1.5.x
    - Positionner la variable d'environnement JAVA_HOME avec la valeur adéquate ;
    - Rajouter le chemin d'installation du JDK au Path du système d'exploitation.
- WSO2 ESB v1.7
    - L'installation requiert moins de 50Mo d'espace disque ;
    - Télécharger la distribution « Binary Distribution » à l'url suivante : http://wso2.org/download/esb
- Browser : Lorsque WSO2 ESB est démarré, on peut accéder à la console d'administration web à l'url suivante : https://localhost:9444/esb. Il est nécessaire de posséder alors l'un des browsers suivants :
    - Mozilla Firefox 2.0 avec la résolution de 1024x768 est recommandé ;
    - MS Internet Explorer 7 peut être aussi utilisé.

3.1.2.2. Installation

- Environnement Linux/Unix

a. Télécharger la distribution « Binary Distribution » de WSO2 ESB ;

b. Extraire l'archive zip dans l'espace d'installation de WSO2 ESB ;

c. Positionner la variable JAVA_HOME en utilisant la commande export ou en éditant le fichier /etc/profile et ajouter le répertoire /bin de Java dans le Path ;

d. Exécuter la commande de démarrage de WSO2 ESB ou le script daemon du répertoire /bin. Par exemple : ./wso2-esb.sh ou ./wso2-esb-daemon.sh start ou ./wso2-esb-daemon.sh console ;

e. Vérifier que l'instance de WSO2 ESB est bien démarrée en allant à l'URL suivante : https://localhost:9444/esb qui affichera la console web conviviale d'administration ;

f. Se logger comme « admin » en utilisant le mot de passe par défaut « admin ».

- Environnement Windows

a. Télécharger la distribution « Binary Distribution » de WSO2 ESB ;

b. Extraire l'archive zip dans l'espace d'installation de WSO2 ESB ;

c. Positionner la variable JAVA_HOME et ajouter le répertoire /bin de Java dans le Path ;

d. Exécuter la commande de démarrage de WSO2 ESB en utilisant la commande wso2-esb.bat ;

e. Si on souhaite l'installer comme un service Windows, utiliser le script : install-wso2-esb-service.bat script ;

f. Vérifier que l'instance de WSO2 ESB est bien démarrée en allant à l'URL suivante : https://localhost:9444/esb qui affichera la console web conviviale d'administration ;

g. Se logger comme « admin » en utilisant le mot de passe par défaut « admin ».

3.2. WSO2 ESB PAR LA PRATIQUE : PROXY SERVICES

3.2.1. DESCRIPTION DE LA FONCTIONNALITE

Comme son nom l'indique, un service proxy agit comme un service hébergé au niveau de l'ESB, WSO2 ESB est alors un frontal au service réel existant. Un service proxy peut être créé et exposé sur un mode de transport différent, un WSDL ou un QOS (Quality Of Servive) du service réel et peut agir comme un médiateur sur les messages du client vers le Web Service ou encore du Web Service vers le client.

Les clients envoient les requêtes vers le service proxy (qui est le seul à connaître la situation des services distants). Ces requêtes sont alors dans un second temps traitées et acheminées vers le service réel s'exécutant sur une machine distante. Ceci dit, il n'est pas toujours nécessaire de toujours réacheminer les messages vers le web service réel. Il est alors possible de lister une combinaison de tâches à effectuer sur les messages reçus par le service proxy, de mettre fin au flux échangé ou d'acheminer un message au client sans même passer par le web service réel. Nous exposerons dans cette section un exemple réalisant la notion de service proxy.

3.2.2. CONFIGURATION DE WSO2 ESB

Démarrer l'ESB en suivant les différentes étapes énumérées dans les sections précédentes. Lancer la console d'administration. Choisir le menu « Proxy Services » dans le panneau gauche. Ajouter un nouveau proxy en choisissant « add ». La page de configuration suivante s'affiche :

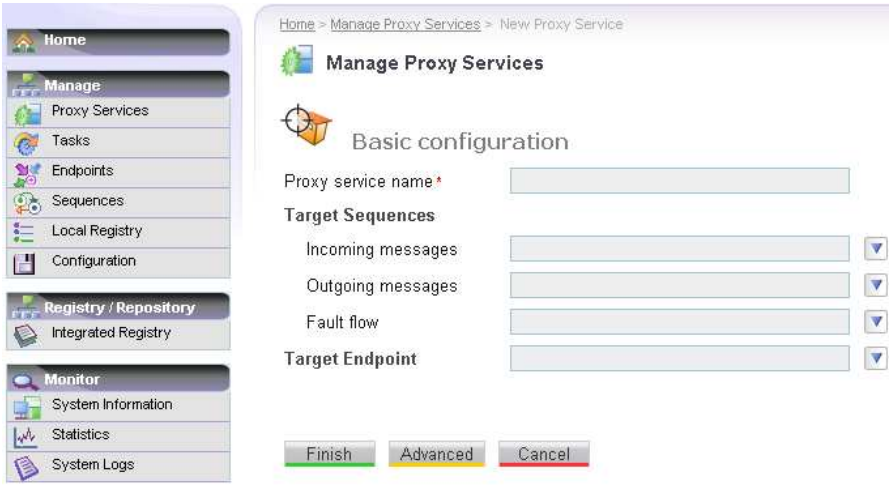

Figure 3 : Configuration d'un WSO2 ESB en mode Service Proxy

Donnez un nom au service proxy à créer. Prenons par exemple un service d'authentification d'un utilisateur dans une application web. Nous aurons ainsi un Web Service distant dénommé UserService. Appelons son service proxy « UserProxy ». Il faudra dans un second temps rajouter une « target endpoint » pour manipuler les requêtes et une « target out sequence » pour les réponses vers le client. Paramétrer le « Target Endpoint », en le choisissant comme « Specify as Anonymous » et en spécifiant l'adresse de l' « endpoint », ici : *http://rnsxp-444:8080/crm-1.0-SNAPSHOT/services/UserService*.

Après avoir effectué les paramétrages avancés :

- les options de configuration du mode de transport : le service proxy est par défaut exposé à la fois sur les protocoles de transport http et https ;
- La configuration des QOS (Quality Of Service) comme la sécurité ;
- Spécification du WSDL (du web service réel) pour le service proxy.

Le nouveau service Proxy « UserProxy » est alors créé. On peut alors activer les options concernant les statistiques et les traces en cliquant sur les icônes appropriées.

Figure 4 : Création d'un Service Proxy nommé UserProxy

3.2.3. CONSOMMATION ET GENERATION DE STATISTIQUES ET DE TRACES

Après l'appel du client, en soumettant sa requête directement au « end point » de transport : http://localhost:8280/soap/UserProxy, il est alors possible de visualiser les statistiques générées par le WSO2 ESB, de même que les traces de l'appel. Nous notons sur la Figure 5 que 2 messages sont passés au travers du service proxy UserProxy. Des informations plus détaillées concernant les statistiques du service proxy sont accessibles en cliquant sur le lien : « Proxy Service Statistics Summary ».

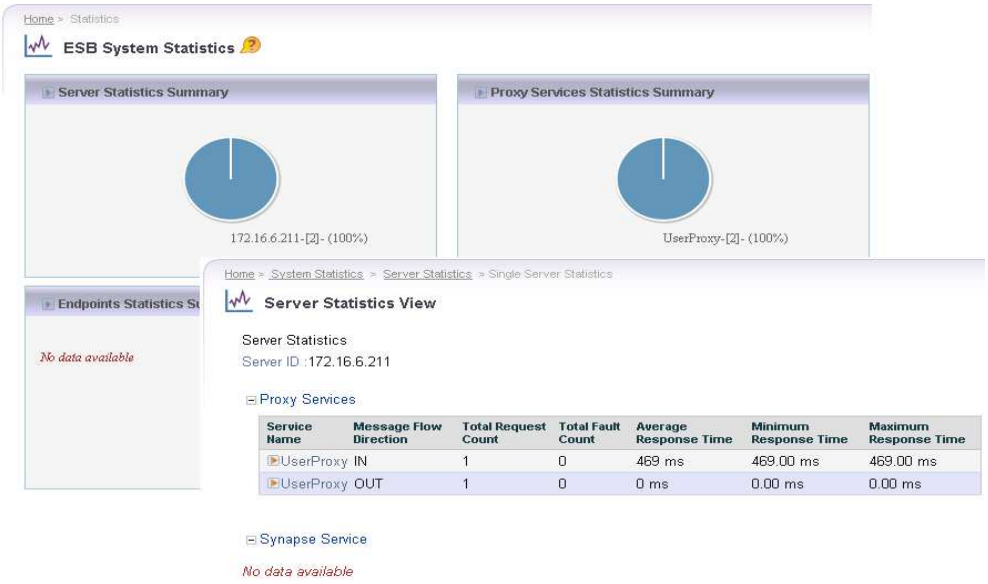

Figure 5 : Génération de statistiques par le WSO2 ESB

Il est aussi possible d'analyser les interactions entre les différentes entités avec le WSO2 ESB au travers de la trace qu'il génère.

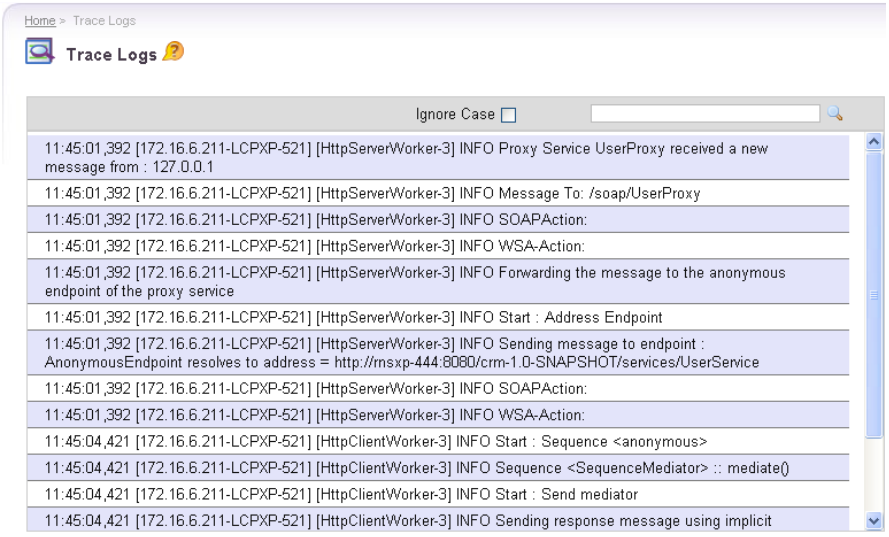

Figure 6 : Génération de traces par le WSO2 ESB

Le but de ce document n'est pas d'expérimenter les différentes fonctionnalités offertes par WSO2 ESB. Ces fonctionnalités (répartition de charges, fail-over, load-balancing, service discovery…) doivent être cependant implémentées pour gérer des systèmes complexes. Cette étude va s'appesantir sur une implémentation de la sécurité des web services au travers de WSO2 ESB. Quels sont les différents paramètres nécessaires à la mise en œuvre d'une telle sécurisation ? Quelles sont les limites pour une architecture comme celle dont nous faisons l'étude?

3.3. WSO2 ESB ET LA SECURITE

3.3.1. WS-SECURITY : RAMPART

WS-Security (Web Services Security) est un protocole de communication qui permet d'appliquer de la sécurité aux Web Services. Il permet d'inclure l'ensemble des systèmes nécessaires à la mise en œuvre d'une transaction sécurisée entre une application et un Web Service. A cette fin, WS-Security enrichit le protocole SOAP de plusieurs nouveaux objets, qui s'associent pour transmettre un jeton de sécurité entre le client et le serveur. Le jeton de sécurité peut être fort simple (mot de passe crypté) ou très compliqué (certificat numérique). Son rôle est de protéger le cœur du message SOAP, en laissant les autres parties du message ouvertes à l'inspection par les pare-feux, les passerelles, et les routeurs de Web Services.

Apache Rampart/Java est une implémentation du standard WS-Security pour le moteur des web services : Axis2, basé sur Apache WSS4J et les implémentations Apache AXIOM-DOOM. Il s'agit d'un produit de la Fondation Apache : *Apache Software Foundation*. WSO2 ESB est sécurisé par le biais d'Apache Rampart. Il a été conçu pour fonctionner avec Axis2/Java.

3.3.2. WS-POLICY : BREF APERÇU

WS-Policy est une grammaire flexible et extensible qui permet d'exprimer les possibilités, exigences et caractéristiques générales d'entités dans un système basé sur les services web XML. WS-Policy définit un cadre et un modèle pour l'expression de ces propriétés en tant que « policies ».

Il représente ainsi un ensemble de spécifications qui décrit les possibilités et les contraintes des « policies » de sécurité (et d'autres domaines) sur les intermédiaires et les « end point » par exemple, (gages sur la sécurité, algorithmes de chiffrement supportés et règles sur la vie privée) et comment associer des « policies » à des services et des « end point ».

En particulier, WS-policy est une spécification qui permet :

- Aux consommateurs de services web de spécifier leurs exigences en matière de « policy » ;
- Aux services web d'utiliser XML pour annoncer leurs « policies » (sur la sécurité, la qualité de service, etc.).

3.3.3. CAS PRATIQUE

Les Web Services présentent l'avantage d'être plus simple à développer que les EJBs (Enterprise Java Bean), néanmoins certains aspects comme la sécurité peuvent devenir vraiment complexes. Dans la plupart des cas, il faut pouvoir découpler le développement du Web Service et la notion de sécurité. La sécurité doit être vue comme de l'intégration et non comme du développement logiciel, elle doit être transverse, c'est à dire qu'ajouter de la sécurité pour un Web Service ne doit pas nécessiter un redéveloppement du Web Service. L'idée serait alors de gérer cette problématique via un ESB (WSO2 ESB dans notre cas). WSO2 ESB permet facilement de gérer la sécurité en mode proxy. Le cas pratique suivant s'attachera à montrer un exemple de sécurisation des Web Services à l'aide de WSO2 ESB.

La Figure 7 montre de façon sommaire les différentes interactions entres le service appelant (le client ou le consommateur), l'ESB (le service proxy, les transformations..) et enfin le service exposé.

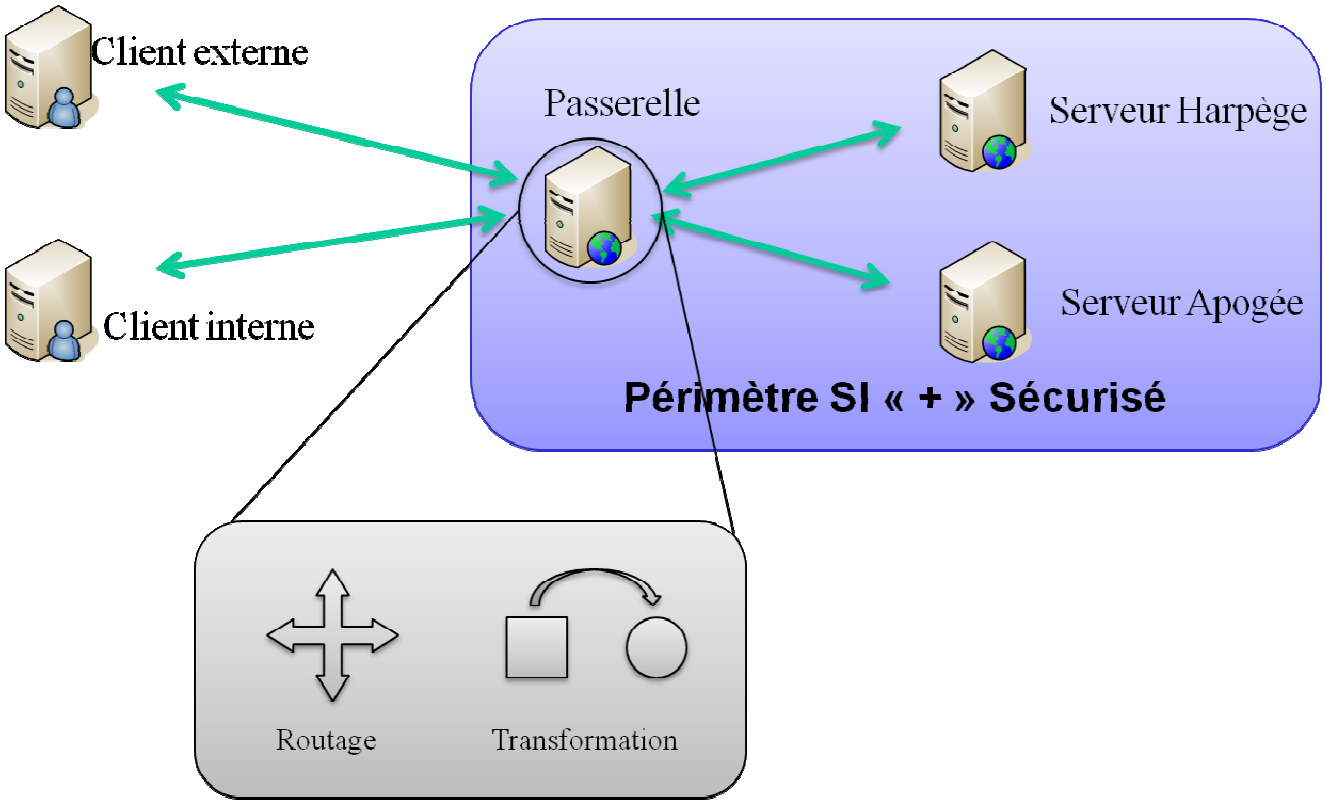

Figure 7 : Interactions entre le consommateur, le WSO2 ESB et le fournisseur

Le cas pratique de sécurisation que nous aborderons est le suivant :

- Le client crée une requête sous forme de message SOAP. Ce message est signé et encrypté puis envoyé au service proxy de l'ESB ;
- Le message est envoyé alors sur le réseau, transporté par http ;
- L'ESB a été configuré pour ne recevoir que des messages signés et encryptés pour le service proxy visé. Il reçoit le message SOAP et en vérifie l'entête :
    - S'il s'agit d'un message signé et encrypté, il se sert des paramètres du service proxy, sis dans le fichier de « policy » (exposé plus loin dans ce document), pour en ôter l'entête WS-Security du message ;
    - S'il s'agit d'un message non signé et non encrypté à destination du service Proxy, alors une erreur est renvoyée au client.
- Le message en clair est alors acheminé jusqu'au Web Service réel. Ce dernier fourni le service escompté et renvoi une réponse en clair à l'ESB ;
- L'ESB se charge de signer et d'encrypter la réponse à l'aide des paramètres contenus dans le fichier de « policy » et la renvoie au client.
- Le client décrypte le message par le biais du module Rampart et en récupère le contenu.

3.3.3.1. Mise en place de WSO2 ESB en mode proxy

Dans un premier temps, il faut pouvoir mettre en place WSO2 ESB en mode proxy au dessus d'un serveur Axis 2 existant par exemple. Cette mise en place se fait de la façon suivante dans le fichier %WSO2_HOME%/conf/synapse.xml. Notons ici que WSO2 ESB permet aussi bien de paramétrer ces différents éléments depuis sa console d'administration web, mais permet aussi de charger une configuration particulière lors de son démarrage, et ce par le biais d'un fichier xml dénommé synapse.xml. Ceci en facilite d'avantage l'utilisation.

```
<!-- Using WS-Security with policy attachments for proxy services -->
<definitions xmlns="http://ws.apache.org/ns/synapse">

<!-- <localEntry> précise le chemin du fichier de définition des règles de sécurité -->
<localEntry key="sec_policy" src="file:repository/conf/resources/policy/policy.xml"/>
<!-- <proxy> précise le nom du service proxy et ses paramétrages-->
    <proxy name="StockQuoteProxy">
        <target>
                <!-- <inSequence> on précise ici les actions à mener sur les messages entrants-->
            <inSequence>

                <!-- <header>les en-têtes de sécurité seront enlevés des messages encryptés-->
                 <header name="wsse:Security" action="remove"
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                        wssecurity-secext-1.0.xsd"/>
                 <send>
                <!-- <endpoint> précise la destination des messages sortant vers le web service réel-->
                     <endpoint>
                         <address uri="http://localhost:9000/soap/SimpleStockQuoteService"/>
                     </endpoint>
                 </send>
            </inSequence>
                <!-- <outSequence> on précise ici les actions à mener sur les messages sortant-->
                <outSequence>
                        <send/>
                </outSequence>
        </target>
        <!-- <publishWSDL> le chemin au fichier WSDL-->
        <publishWSDL uri="file:repository/conf/sample/resources/proxy/sample_proxy_1.wsdl"/>
        <!-- <policy> indique que le médiateur de gestion des politiques de sécurité est initialisé-->
        <policy key="sec_policy"/>
        <!-- <enablesec> activation la gestion de la sécurité-->
        <enableSec/>
    </proxy>
</definitions>
```

Listing 3.3.3.1-1 : Le fichier synapse.xml

Démarrer alors WSO2 ESB en lançant le script % WSO2 _HOME%/bin/wso2-esb.bat

3.3.3.2. Création d'un entrepôt de clés : le keystore

L'outil keytool fournit par Sun permet de gérer des entrepôts de clés privées. Il est préférable d'utiliser l'algorithme RSA car certaines librairies java ne supportent pas DSA. De plus, le hachage est fait par défaut en MD5 qui n'est plus sécurisé. Il faut donc utiliser SHA1 pour la signature. Le certificat issu de cet exemple sera auto-signé. Il est préférable d'utiliser un certificat signé par un tiers de confiance.

→ création d'un entrepôt de clés nommé store.jks avec un alias myUser, un alogorithme RSA, une signature SHA

```
"%JAVA_HOME%\bin\keytool.exe" -genkey -keystore store.jks -alias myUser -keyalg RSA -sigalg
SHA1withRSA
Tapez le mot de passe du Keystore :  mykeystore
Quels sont vos prénom et nom ?
  [Unknown] :
```

```
Quel est le nom de votre unité organisationnelle ?
  [Unknown] :
Quelle est le nom de votre organisation ?
  [Unknown] :
Quel est le nom de votre ville de résidence ?
  [Unknown] :
Quel est le nom de votre état ou province ?
  [Unknown] :
Quel est le code de pays à deux lettres pour cette unité ?
  [Unknown] :
Est-ce CN=myFirstName mySecondName, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown ?
  [non] :  oui
Spécifiez le mot de passe de la clé pour <myUser>
        (appuyez sur Entrée s'il s'agit du mot de passe du Keystore) :  myPassword
```

Listing 3.3.3.2-1 : La création du keystore

On a donc généré un keystore « store.jks » sécurisé par le mot de passe « *myKeystore* » avec une nouvelle entrée (alias « *myUser* ») correspondant à une nouvelle paire de clés privée / publique. La clé publique est encapsulée dans un **certificat X509** pour l'utilisateur « *myUser* » avec comme mot de passe « *myPassword* ».

3.3.3.3. Définition des politiques de sécurité avec WS-Policy

Un client de Web Service ne peut pas accéder à un Web Service si celui ci a défini des règles demandant à ce que les appels soient cryptés et signés d'une certaine façon. WS-Policy comme nous l'avions abordé plus haut dans ce document, permet de définir les règles de consommation d'un Web Service par un client. WS-Policy ajoute ainsi une sécurité supplémentaire à WS-Security car un client de Web Service ne peut pas appeler un Web Service sans respecter les règles de sécurité établies dans la politique WS-Security (WS-SecurityPolicy).

Pour définir une politique de sécurité pour les Web Services dans WSO2 ESB il est nécessaire de créer un fichier policy.xml dans %SYNAPSE_HOME%/repository/conf/resources/policy avec le contenu suivant:

```
<wsp:Policy     wsu:Id="SigEncr"     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">

        <wsp:ExactlyOne>
           <wsp:All>
                <!-- <sp :AsymetricBinding>permet de définir une sécurité de type X509 contenue dans le message
                contrairement à <sp :TransportBinding> qui définit une sécurité basé sur le transport -->
                <sp:AsymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">

                      <wsp:Policy>
                             <!-- <sp :InitiatorToken> permet de définir le type de jeton pour le client du web service--
                      >
                             <sp:InitiatorToken>
                                    <wsp:Policy>
                                           <sp:X509Token
                                           sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005
                                           /07/securitypolicy/IncludeToken/AlwaysToRecipient">
                                                  <wsp:Policy>
                                                         <sp:WssX509V3Token10/>
                                                  </wsp:Policy>
                                           </sp:X509Token>
                                    </wsp:Policy>
                             </sp:InitiatorToken>

                             <!-- <sp :RecipientToken> permet de définir le type de jeton pour le web service-->
                             <sp:RecipientToken>
                                    <wsp:Policy>
```

```
                                <sp:X509Token
                                sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005
                                /07/securitypolicy/IncludeToken/Never">
                                        <wsp:Policy>
                                                <sp:WssX509V3Token10/>
                                        </wsp:Policy>
                                </sp:X509Token>
                        </wsp:Policy>
                </sp:RecipientToken>

                <!-- <sp :AlgorithmSuite> permet de définir le type d'algorithme requis-->
                <sp:AlgorithmSuite>
                        <wsp:Policy>
                                <!-- <sp :Basic256>Type d'algorithme : RSA 256 bits et SHA1-->
                                <sp:Basic256/>
                        </wsp:Policy>
                </sp:AlgorithmSuite>

                <sp:Layout>
                        <wsp:Policy>
                                <!-- <sp :Strict> Indique que les règles sont ordonnées-->
                                <sp:Strict/>
                        </wsp:Policy>
                </sp:Layout>

                <!-- <sp :IncludeTimestamp>Indique qu'un timestamp sur le message est requis-->
                <sp:IncludeTimestamp/>

                <!-- <sp :OnlySignEntireHeadersAndBody>Indique que la signature se fait sur l'en-tête et
                le corps du message uniquement-->
                <sp:OnlySignEntireHeadersAndBody/>
        </wsp:Policy>
</sp:AsymmetricBinding>

<sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <wsp:Policy>
                <sp:MustSupportRefKeyIdentifier/>
                <sp:MustSupportRefIssuerSerial/>
        </wsp:Policy>
</sp:Wss10>
<!-- <sp :SignedParts>on indique que le body est signé-->
<sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <sp:Body/>
</sp:SignedParts>
<!-- <sp :EncryptedParts>on indique que le body est crypté-->
<sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <sp:Body/>
</sp:EncryptedParts>
<!-- <sp :RampartConfig>on indique la configuration Rampart-->
<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
        <ramp:user>myUser</ramp:user>
        <ramp:encryptionUser>myUSer</ramp:encryptionUser>
        <!-- <ramp :passwordCallbackClass >on indique la classe du callback-->
        <ramp:passwordCallbackClass>samples.userguide.PWCallback
        </ramp:passwordCallbackClass>

        <!-- <ramp :signatureCrypto >on indique les paramètres rampart pour la signature-->
        <ramp:signatureCrypto>
                <ramp:crypto provider="org.apache.ws.security.components.crypto.Merlin">

                        <ramp:property  name="org.apache.ws.security.crypto.merlin.keystore
                        .type">JKS</ramp:property>

                        <ramp:property    name="org.apache.ws.security.crypto.merlin.file">
                        repository/conf/resources/security/store.jks</ramp:property>

                        <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.
                        password">myKeystore</ramp:property>
                </ramp:crypto>
        </ramp:signatureCrypto>

        <!-- <ramp :encryptionCypto >on indique les paramètres rampart pour l'encryption-->
        <ramp:encryptionCrypto>
                <ramp:cypto provider="org.apache.ws.security.components.crypto.Merlin">
                        <ramp:property  name="org.apache.ws.security.crypto.merlin.keystore
                        .type">JKS</ramp:property>
```

```
                                    <ramp:property    name="org.apache.ws.security.crypto.merlin.file">
                                    repository/conf/resources/security/store.jks</ramp:property>

                                    <ramp:property  name="org.apache.ws.security.crypto.merlin.keystore
                                    .password">mKeystore</ramp:property>
                                </ramp:crypto>
                            </ramp:encryptionCypto>
                        </ramp:RampartConfig>

                    </wsp:All>
                </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 3.3.3.3-1 : Le fichier de policies : policy.xml

3.3.3.4. Mise en place de WS-Security sur WSO2 ESB

Ajout du code permettant la gestion des mots de passe :

Créer la classe « PWCallback » et déployer le jar (`jar cf NOM_DU_JAR FICHIER_A_ARCHIVER` pour creer un jar et `jar xf NOM_DU_JAR FICHIER_A_ARCHIVER` pour extraire un jar) contenant cette classe dans le répertoire %SYNAPSE_HOME%/lib:

```
//on indique les imports
import org.apache.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PWCallback implements CallbackHandler {

 public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        //on parcourt la liste des callbacks
        for (int i = 0; i < callbacks.length; i++) {
                if (callbacks[i] instanceof WSPasswordCallback) {
                        //le callback actuel est du type WSPasswordCallback
                         WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
                        //Si l'identifiant est myUser, alors on insert le mot de passe myPassword
                        if (pc.getIdentifer().equals("myUser")) {
                                pc.setPassword("myPassword");
                        } else {
                        //sinon on génère une exception
                                throw   new   IllegalStateException("Unrecognized   Identifier:"   +
                                pc.getIdentifer());
                        }
                } else {
                        throw    new    UnsupportedCallbackException(callbacks[i],    "Unrecognized
                        Callback");
                }
        }
 }
}
```

Listing 3.3.3.4-1: La classe PWCallback.java

3.3.3.5. Modifier le client pour activer la sécurité

Ajouter le répertoire où se trouvent les modules « *addressing.mar* » et « *rampart.mar* » en copiant le répertoire modules dans le projet client se trouvant dans %SYNAPSE_HOME%/repository.

Modifier le code source du client comme suit, où le répertoire resources/repository correspond au répertoire où se trouvent les modules (modules/addressing et modules/rempart)

```
Options options = new Options();
OMElement payload = null;
ServiceClient serviceClient;
…
//on indique l'engagement du module addressing
serviceClient.engageModule("addressing");
//on indique l'engagement du module rampart
serviceClient.engageModule("rampart");
//on indique le chemin du fichier de policies
options.setProperty(RampartMessageData.KEY_RAMPART_POLICY,
loadPolicy("./../../repository/conf/resources/policy/policy.xml"));
serviceClient.setOptions(options);
//on envoie la requête
serviceClient.sendReceive(payload);

…
//méthode permettant de charger des polices à partir de neethi, une librairie java facilitant la gestion des policies WS-Policy
private static Policy loadPolicy(String xmlPath) throws Exception {
 StAXOMBuilder builder = new StAXOMBuilder(xmlPath);
 return PolicyEngine.getPolicy(builder.getDocumentElement());
}
```

Listing 3.3.3.5-1: L'appel du consommateur (le client)

3.3.3.6. Les flux échangés au travers de Tcpmon

- Section Client – ESB (Service Proxy) :

La figure suivante expose les différents flux échangés entre le client et le WSO2 ESB. La première fenêtre correspond au message SOAP émis par le client vers l'ESB. L'entête contient les attributs de WS-Security, le message est chiffré. La seconde fenêtre correspond au message SOAP de réponse acheminée de l'ESB vers le client. On y voit aussi les attributs WS-Security, ce qui prouve bien que le message est crypté.

Ici Tcpmon écoute sur le port 8281 (port d'émission du message du client) et retransmet sur le port 8280, port d'écoute de l'ESB.

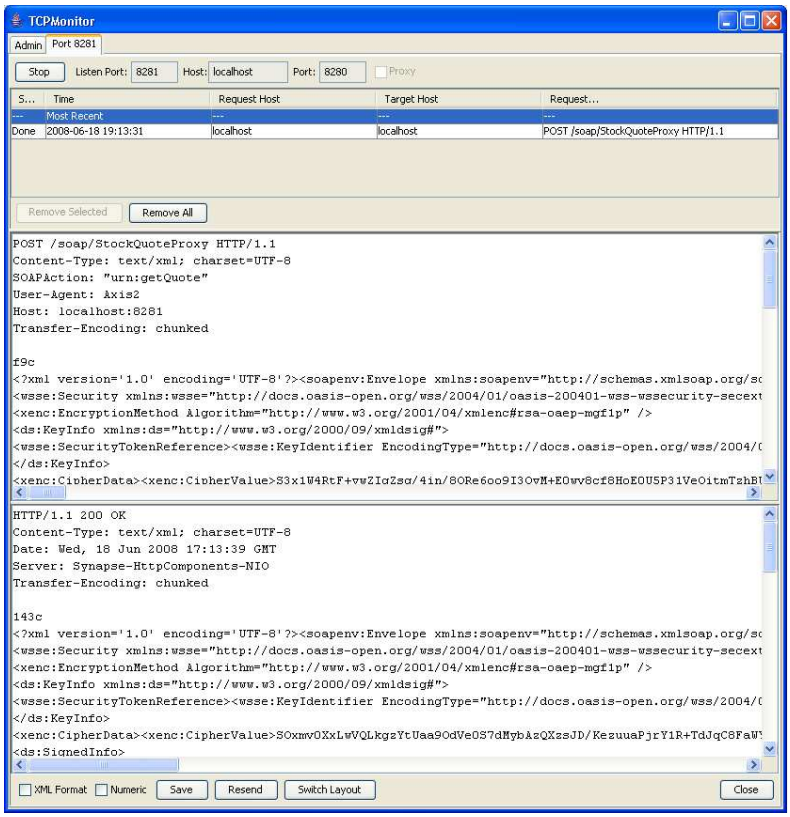

Figure 8 : Capture Tcpmon section consommateur-WSO2 ESB

- Section ESB (Service Proxy) -Web Service :

La figure suivante expose les différents flux échangés entre le WSO2 ESB et le Web Service réel. La première fenêtre correspond au message SOAP émis par l'ESB vers le Web Service. L'entête ne contient aucun attribut de WS-Security. Ils ont été supprimés par l'ESB. Le message n'est pas chiffré. La seconde fenêtre correspond au message SOAP de réponse acheminé du Web Service vers l'ESB. On n'y voit aussi aucun attribut WS-Security, ce qui prouve bien que le message est en clair.

Ici Tcpmon écoute sur le port 9001 (port d'émission du message de l'ESB vers le Web Service) et retransmet sur le port 9000, port d'écoute du Web Service.

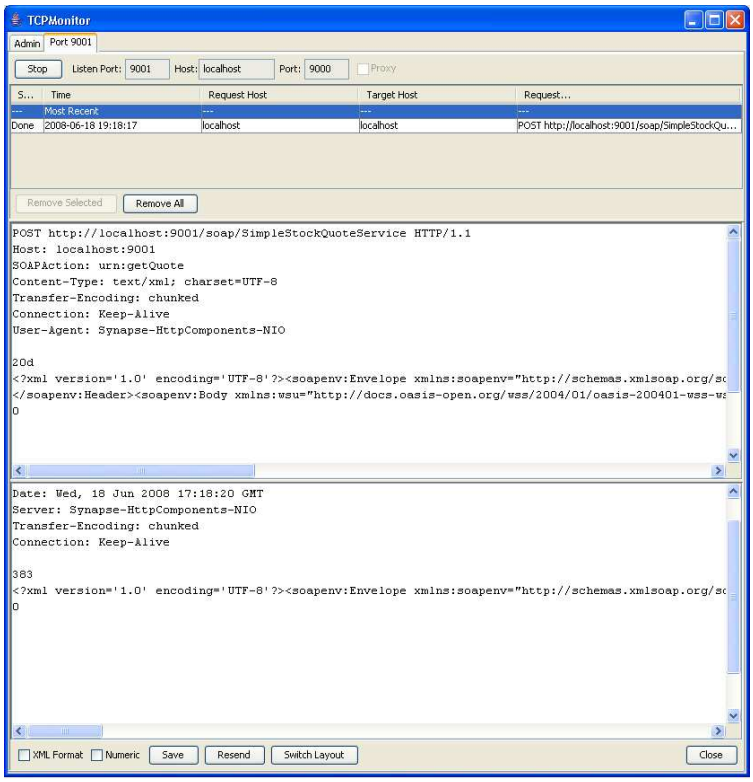

Figure 9 : Capture Tcpmon section WSO2 ESB-Fournisseur

3.3.4. LIMITES

A la lueur des différentes approches effectuées tout au long de ce document, nous en arrivons aux conclusions suivantes : WSO2 ESB permet d'exposer des Web Services en interne ou en externe. Les concepts de SOA n'obligent pas de disposer d'un client écrit dans un langage particulier pour consommer le service. Ainsi, le service peut être codé dans n'importe quel langage, et s'exécuter sur n'importe quelle plateforme (matérielle et logicielle), de même que le client. Ainsi, par exemple, un client C ou même .Net peut consommer un Web Service Java.

D'un point de vue des notions de sécurité de l'étude, à la différence de SPRING SECURITY qui met en place les notions d'authentification et d'autorisation, Axis2/Rampart, bien que permettant l'authentification, ne permet pas d'aborder le cas des autorisations de façon simple et standard. Il est cependant possible de pallier à ce manque en récupérant au niveau du PWCallback, le messageContext, et d'y récupérer le nom du service, de même que la méthode appelée et de vérifier ces informations dans une base de données. Ceci afin d'établir si l'entité appelante dispose de permissions suffisantes afin d'accéder à cette ressource ou non. Des exemples en C existent sur le site de WSO2.

Malgré tout, le groupe de travail a considéré qu'il n'était pas du rôle de l'ESB d'assurer les fonctions d'autorisations qui sont à rapprocher de la logique métier de chaque application.