Implémentation

Dans cette partie, nous présentons un scénario (§VI.1) qui illustre un cas d'usage de la personnalisation des services et qui montre l'utilité d'avoir une autogestion de la QoS, basée sur un agent de QoS dans un contexte ubiquitaire, au cours de la mobilité de la session. Puis dans (§VI.2), nous détaillons notre plate-forme de développement et de tests UBIS, qui est utilisée pour le développement des services applicatifs ainsi que ceux de gestion que nous avons proposé pour gérer la création et la modification de la session UBIS.

VI.1 Scénario

Dans cette section, nous utilisons un cas d'usage pour démontrer l'utilité de l'ensemble de nos propositions. Le scenario suivant illustre le cas où un utilisateur souhaite personnaliser ses services pour chercher un bien immobilier à acheter dans sa zone géographique.

Il choisit dans son catalogue trois services exposables dont il a besoin : Search Proprety, Google Maps , et banking.

Services exposables	Recherche bien immobilier (Search proprety)			Google Maps		Banque (Banking)		
Services de base	FIND	LOCATION	GET	Display on Map	Calculate Itiniraire	Demand credit	Insurance	Secure Payment
Qualité de service	Haute			Haute		Haute		
Fournisseur	Square Habitat			Google		Crédit agricole		

Le scénario se déroule comme suit (Figure 39):

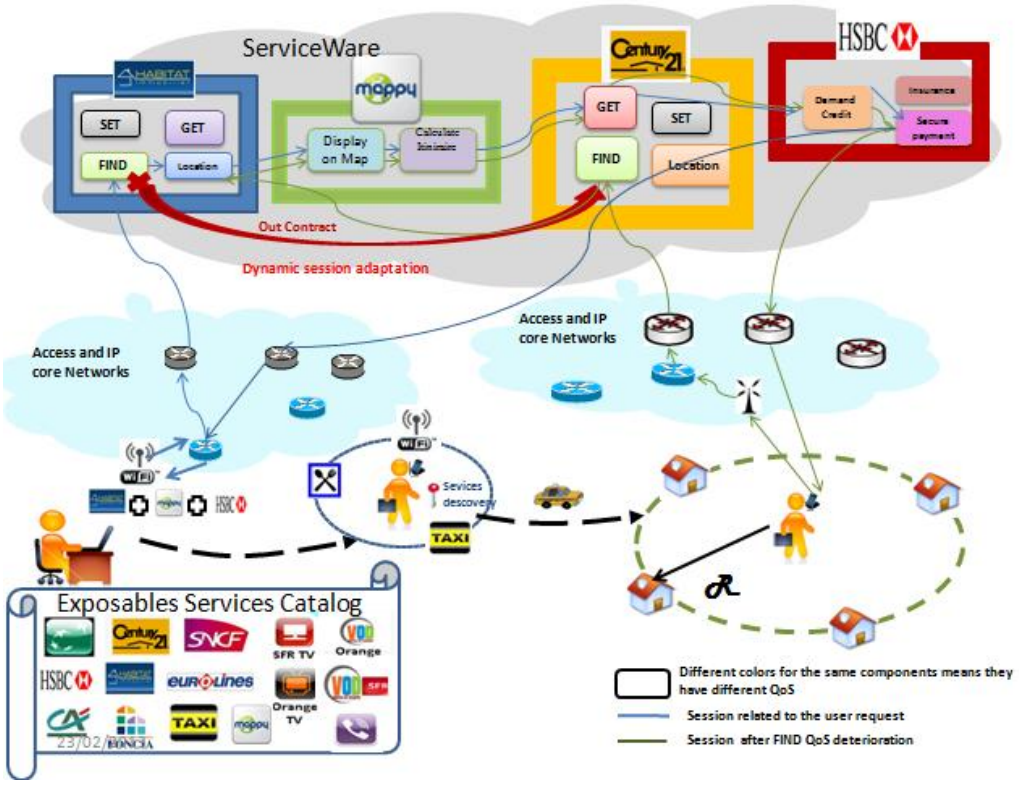

Figure 39 : Scénario de la mobilité de la session des services personnalisés

Lorsque l'utilisateur est en train de se déplacer, il va utiliser le composant FIND pour rechercher des biens immobiliers en fonction de sa position géographique (Longitude, Latitude). Cette recherche est effectuée dans la zone ambiante de l'utilisateur qui est limitée par un rayon R. Ensuite le composant de service «Location» est utilisé pour obtenir les coordonnées GPS pour chaque bien immobilier. Ensuite il utilise Google Maps pour localiser le bien sur une carte, et après il utilise le composant de service «GET» pour voir le profil détaillé de chaque bien sélectionné (la nature, la superficie, le prix, l'adresse). Et finalement, il va utiliser le service «Banking» pour d'une part demander un prêt d'achat et une assurance pour le bien qu'il a choisi, et d'autre part pour effectuer, le paiement sécurisé en ligne.

Au cours de l'exploitation, la performance de " FIND@Habitat" se détériore et ne peut pas fournir la qualité de service requise par l'utilisateur. Par conséquent, "FIND@Habitat" doit être remplacé par un composant de service FIND ubiquitaire. FIND@HABITAT envoie un

évènement de notification «Out contrat» via le message SIP+ Notify à tous les membres de sa communauté VSC pour traiter son changement. Il existe toujours des composants ubiquitaires fournis par d'autres fournisseurs tels que FIND@Century21.

Pour tester la performance de nos propositions, nous avons fait tourner ce scénario dans un premier temps avec des composants de service sans l'agent QoS (BSC), et dans un second temps avec des composants de service autonomiques (ASC).

- 1er cas: nous surchargeons le composant de service FIND, on remarque que des données sont perdues à cause de la dégradation de la QoS et par conséquent la session est interrompue.

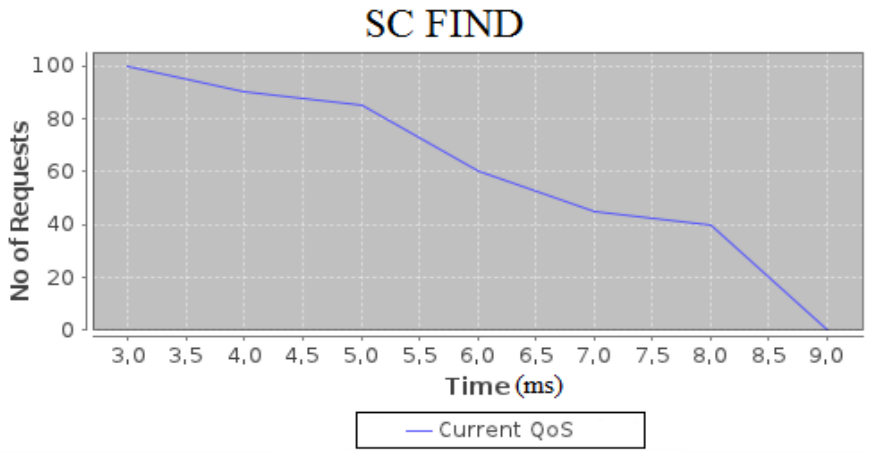

Figure 40 : les performances d'un composant de service basique FIND

- 2ème cas: nous utilisons des composants de service autonomiques ASCs avec les caractéristiques suivantes:

- Find (QoS α): A1= 0, 99, R1= 0, 98, D1=300ms, C1=10.
- Location (QoS β): A2=0, 97, R2=0, 86, D2=500ms, C2=8.
- Get (QoS µ): A4=0, 98, R4=0, 90, D4= 450ms, C4=9.

Nous mentionnons que si un des quatre critères de QoS se dégrade au cours de la session de l'utilisateur, il faut procéder au changement du composant de service parce que tous les critères de QoS sont nécessaires pour maintenir la QoS globale du composant de service.

Durant la session de l'utilisateur, l'agent QoS du composant de service ASC_FIND@Habitat détecte une dégradation de la QoS dans le critère de la Fiabilité (R1min = 0, 8 à T=8ms). Il invoque la communauté VSC pour procéder à son changement par un composant de service équivalent ASC_FIND@Century21.

La performance de la QoS est meilleure grâce au QoS-agent parce qu'il évite une dégradation complète de la QoS d'un composant de service et maintient le «service delivery»

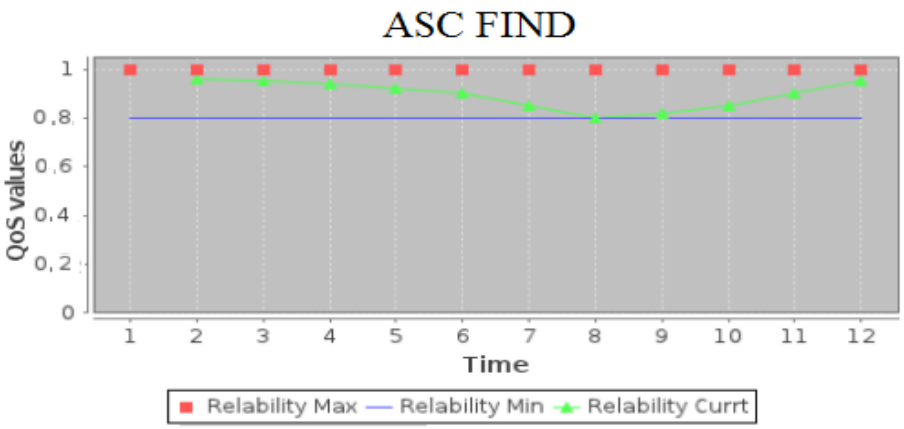

Figure 41: les performances d'un composant de service autonomique FIND

VI.2 Plate-forme de développement et de test UBIS

Pour mettre en place le projet UBIS, nous disposons d'une plate-forme UBIS qui nous permet de démontrer la faisabilité de l'ensemble de nos propositions et les tests des cas d'usages. La plate-forme est développée selon la structure de l'architecture UBIS, elle est structurée en fonctions des quatre niveaux de visibilité: Utilisateur, Terminal, Réseau et Service.

- *Utilisateur:* les abonnements et les préférences de l'utilisateur en termes de personnalisation de service et de mobilité seront stockés dans les bases de connaissances Infosphère et Infoware.
- *Terminal* : l'interface USERWARE fournit les fonctionnalités et les capacités pour permettre à l'utilisateur de personnaliser ses services, et lui offre un accès orienté usage qui est transparent de bout-en-bout aux services demandés.
- *Réseau Accès*: le réseau 3G de la plate-forme R&D de SFR (Société Française de Radiophonie) et le réseau WIFI de Telecom ParisTech sont utilisés pour assurer l'accès aux services.
- *Réseau de transport* : nous nous appuyons sur la solution VIRTUOR qui propose des machines physiques supportant plusieurs entités virtuelles de différents types comme les routeurs IPv4, IPv6, point d'accès virtuel, serveur SIP etc. Cette solution offre la possibilité de mettre en place plusieurs réseaux virtuels, spécifiques aux applications supportées, qui sont vus de la part des utilisateurs comme des réseaux physiques distincts.
- *Plan de contrôle* : nous utilisons le OpenIMS de FOKUS [47] pour mettre en place l'architecture IMS.
- *Les informations* : nous avons une base de connaissance INFOWARE [49] du coté fournisseur et INFOSPHERE [48] du coté utilisateur/terminal.
- *Les services* : nous avons SERVICEWARE, qui représente une architecture de service qui héberge différents types de service : des services applicatifs (Web, ASC) et des services de gestion (MSG).

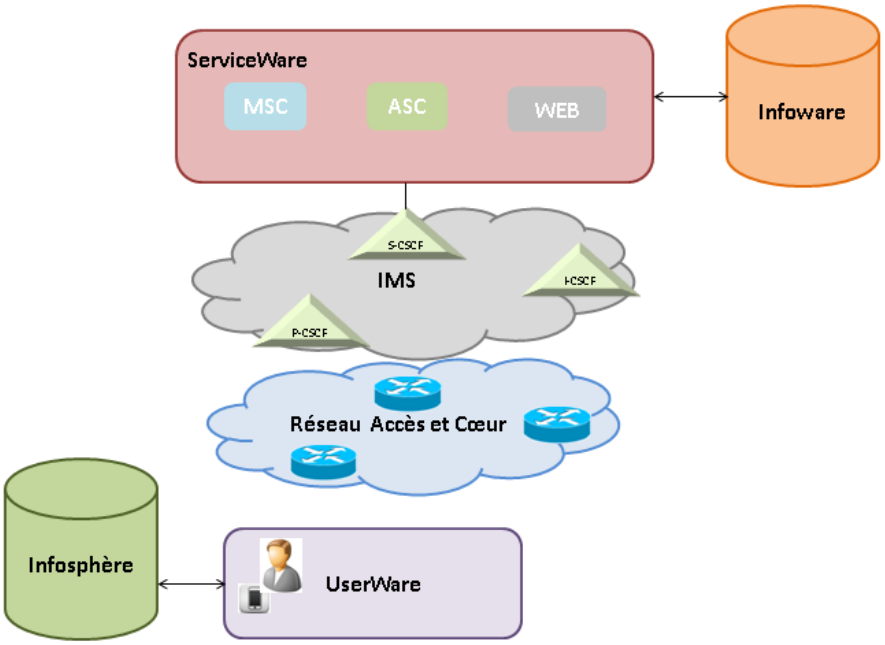

Figure 42: Plate-forme UBIS

VI.2.1 Architecture ServiceWare

Pour mettre en place l'architecture du ServiceWare, nous avons utilisé un service d'application GlassFish V3 /Sailfin pour développer et déployer les services UBIS qui s'appuient sur l'architecture SOA. Nous avons utilisé le langage Java pour développer les composants de service de type gestion et applicatif comme des EJBs indépendants. Il faut mentionner qu'EJB3.0 permet le développement des composants de services autonomes avec un couplage lâche. Ces composants sont déployés dans un serveur Sailfin certifié JEE6, qui supporte différentes APIs comme JMS, JNDI et JDBC et SIP servlet.

Pour les besoins de signalisation de la session et d'usage, les requêtes du protocole SIP + sont créées avec le client SIPP [46] pour créer, modifier et terminer les sessions de services personnalisés, et les requêtes HTTP sont utilisées pour la consommation des services. Nous avons une interface convergée SIP/HTTP «JSR Converged Applications» qui représente un ensemble de Sip Servlet et Http Servlet qui sont gérés par le même conteneur pour assurer l'interfaçage des différents composants EJB ou Web Services.

Le JNDI (Java Naming and Directory Interface) permet la connexion à des annuaires comme LDAP. Il est utilisé pour assurer la gestion des noms des files d'attente rattachée à chaque composant de service. Le JNDI offre les fonctionnalités de nommage des applications développées en JAVA et assure l'association entre les noms et les objets.

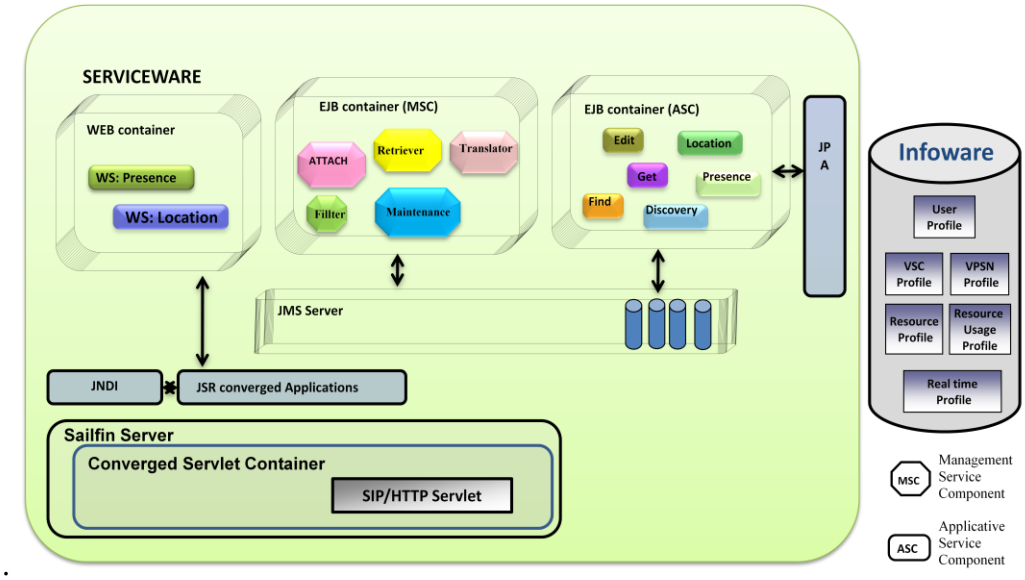

Figure 43: Architecture Serviceware

VI.2.2 La logique de fonctionnement des composants de services de gestion

Comme nous l'avons présenté dans le (§V.3.2), le protocole SIP+ permet le transport des informations fonctionnelles et non-fonctionnelles relatives à la logique des services personnalisés par l'utilisateur. Nous présentons, le modèle fonctionnel et le modèle informationnel, utilisés pour mettre en place les composants de service de types gestion (SIP_Retreiver, Translator, Filter, Attach_VPSN, Maintenance_VPSN) qui sont utilisés pour traiter la demande de l'utilisateur et créer son VPSN.

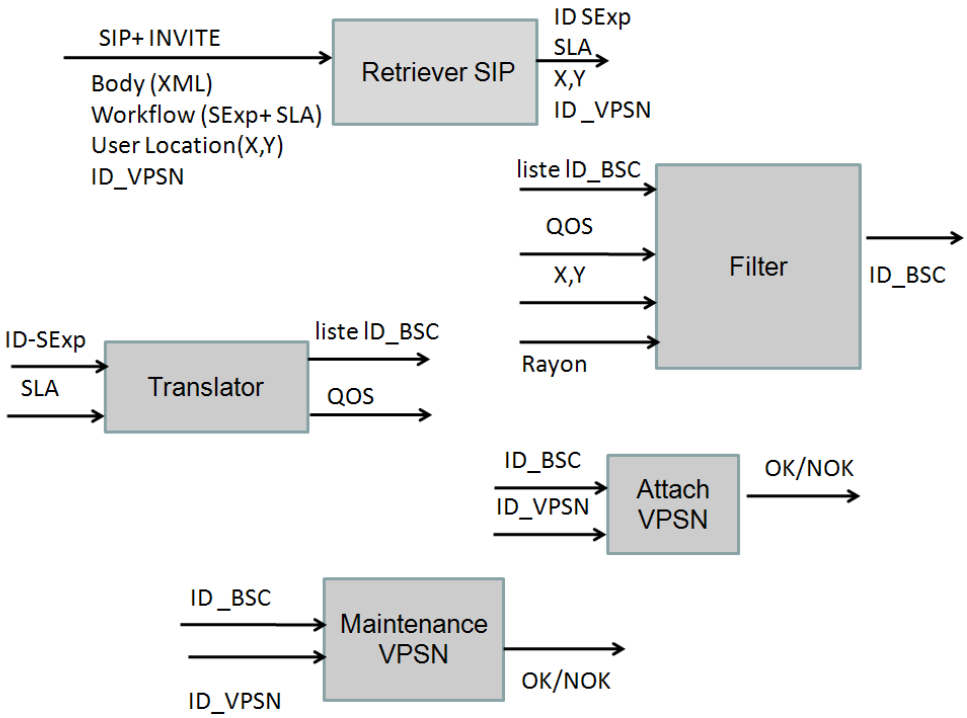

Figure 44 : Modèle fonctionnel des composants de service de gestion

Pour représenter le fonctionnement des différents composants de gestion, nous avons rédigé un pseudo code qui détaille le rôle de chaque composant dans la gestion de la demande de pré-provisioning du VPSN pour l'utilisateur. Ce pseudo code montre également la chaine d'exécution de chaque service suite à la réception d'une requête SIP+ INVITE de demande de création de session par l'utilisateur.

```
 ----------------------------
| ALGORITHM : Vpsn Creation |
| INPUT     : Invite SIP+   |
| OUTPUT    : VPSN Created  |
 ----------------------------

    Input :Sip+ Invite
        I# (-ID-VPSN-) ,(LA,LO)(-User's Location-)
        µ# (-WorkFlow of SExp-)
    BEGIN:
        Step1:
            Sip+ Invite goes to RETREIVER
        RETREIVER:
            Input  : SIP+ Invite Body = {µ#,(LA,LO),I#}
                BEGIN:
                    retrieve information from sip+ Invite Body
                END
            Output :ID-SExp,SLA,(LA,LO),I#
    Step2:
        µ# goes to TRANSLATOR
        TRANSLATOR:
            Input  :ID-SExp , SLA
                BEGIN :
                    FOREACH
                        ID-SExp translate into BSC
                    ENDFOREACH
                END
            Output :List{ID-BSC(Offered-QoS)}
    TRANSLATOR output goes to FILTER

LOOP:BSC-LIST
Step3:
    FILTER:
        Input  :(LA,LO),ID-BSC,Я // User lookup zone radius
            BEGIN:
                    verify Я
                    find ID-BSC
                    send ID-BSC to ATTACH-VPSN
            END
        Output :ID-BSC + Logical-@
    FILTER output goes to ATTACH-VPSN
    ATTACH-VPSN:
    Input : ID-BSC , Logical-@ , I#
        BEGIN:
                verify Ж-QoS // Statistical QoS
                IF !MAX(Ж-QoS)
                    THEN add I# to BSC-Profile
                    ELSE
                        Return to Step3
                    IF FILTER Return == NULL
                        THEN Я++
                        Return to Step3
                    ENDIF
                ENDIF
        END
    Output : OK
    VPSN-MAINTENANCE:
    Input  : ID-BSC , Logical-@ , I#
        BEGIN:
                add ID-BSC + Logical-@ to VPSN-Profile
        END
    Output : OK
ENDLOOP
VPSN-MAINTENANCE Sends SIP+ 200 OK (VPSN Created) to USER
END
```

Figure 45: Pseudo code de la création du VPSN

La base de connaissance Infoware est centralisée et contient tous les profils des différentes ressources: équipements, réseaux services ainsi que le profil de l'utilisateur. Les tables

suivantes sont utilisées pour tester le fonctionnement de nos différents composants de services (usage et gestion) au cours d'une session VPSN.

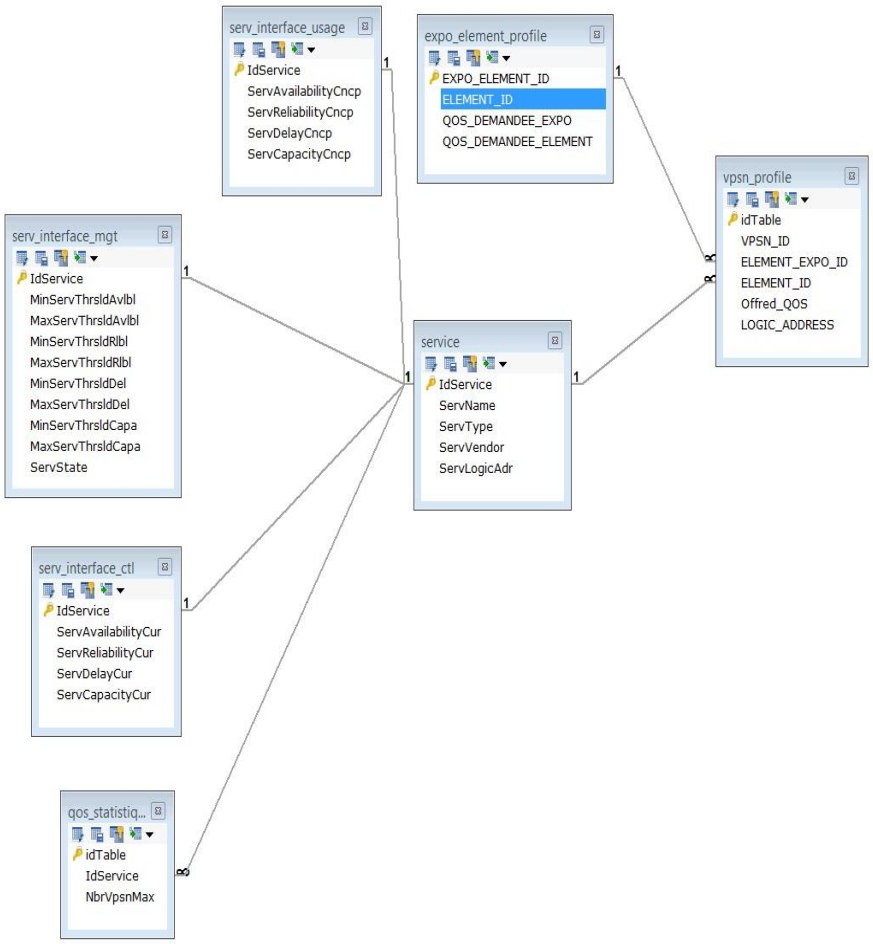

Figure 46: le modèle relationnel des tables du profil des éléments service dans l'Infoware

VI.2.3 Les résultats d'exécution des composants de service de gestion

Dès la réception de l'invite SIP+, le SIP_Retriever procède à la lecture du Contenu XML et extrait les données suivantes

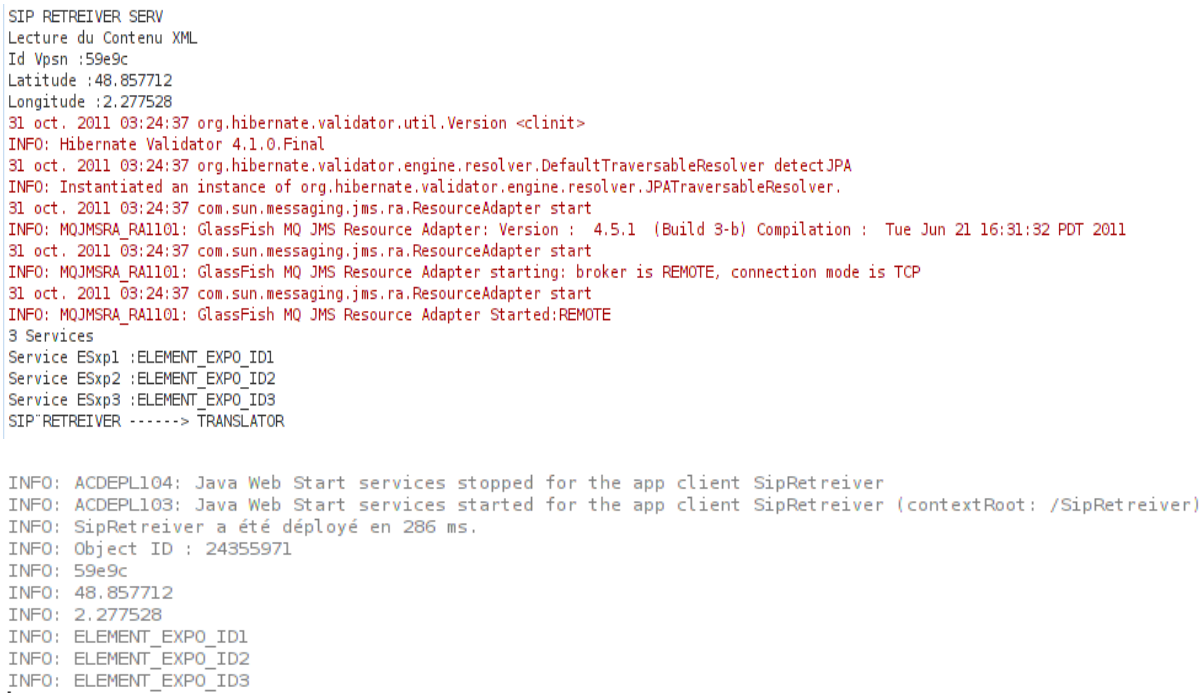

Figure 47: capture d'écran de l'exécution du SIP_Retriever

Le service SIP_Retriever a extrait les données «ID-VPSN», Longitude, Latitude et les IDs de services exposables**.** Ces informations sont envoyées avec des messages objets à la file d'attente JMS et elles sont stockées en FIFO (First In First Out) pour qu'ils soient traités par la suite par le composant Translator.

Chaque service exposable est retiré de la file JMS selon la logique FIFO et traduit par le composant Translator en composants de services élémentaires. Chaque service de base est envoyé conjointement avec sa QoS conceptuelle, l'ID-VPSN et les informations sur la localisation (Lo, La) à la file d'attente du composant de service Filter:

```
------- Translator Serv -------
31 oct. 2011 03:43:12 org.hibernate.validator.util.Version <clinit>
INFO: Hibernate Validator 4.1.0.Final
31 oct. 2011 03:43:13 org.hibernate.validator.engine.resolver.DefaultTraversableResolver detectJPA
INFO: Instantiated an instance of org.hibernate.validator.engine.resolver.JPATraversableResolver.
31 oct. 2011 03:43:13 com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version :  4.5.1  (Build 3-b) Compilation :  Tue Jun 21 16:31:32 PDT 2011
31 oct. 2011 03:43:13 com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
31 oct. 2011 03:43:13 com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Object Found :::
sipretreiver.MyObject@74187a
*******************************
ELEMENT_EXPO_ID1
ELEMENT_EXPO_ID2
ELEMENT_EXPO_ID3
 Element Complexe 1 :ELEMENT_EXPO_ID1
--Element Basique :ELEMENT_ID1.A / [0.9, 0.9, 2500, 100]
--Element Basique :ELEMENT_ID2.A / [0.85, 0.9, 3000, 60]
 Element Complexe 2 :ELEMENT_EXPO_ID2
--Element Basique :ELEMENT_ID2.A / [0.85, 0.9, 3000, 60]
--Element Basique :ELEMENT_ID3.A / [0.8, 0.8, 3100, 120]
--Element Basique :ELEMENT_ID4.A / [0.9, 0.85, 2500, 80]
 Element Complexe 3 :ELEMENT_EXPO_ID3
--Element Basique :ELEMENT_ID1.B / [0.9, 0.89, 2500, 100]
--Element Basique :ELEMENT_ID5.B / [0.9, 0.87, 2500, 100]
Translator ------> Filter
```

Figure 48 : Exécution du Service Translator

Les messages objets envoyés par Translator sont consommés un par un selon une boucle par le composant Filter

```
Object Found :::
translatorcon.MyExpoObject@10f7ec9
*******************************
Element trouvé dans le rayon :ELEMENT_ID2.A
Qos :[0.85, 0.9, 3000, 60]
Adresse logique :sip:ELEMENT_ID2.A@ubis01.sfr.com

Element trouvé dans le rayon :ELEMENT_ID3.A
Qos :[0.8, 0.8, 3100, 120]
Adresse logique :sip:ELEMENT_ID3.A@ubis01.sfr.com

Element trouvé dans le rayon :ELEMENT_ID4.A
Qos :[0.9, 0.85, 2500, 80]
Adresse logique :sip:ELEMENT_ID3.A@ubis01.sfr.com


FILTER --------> ATTACH-VPSN
```

Figure 49: Exécution du Composant Filter

Le composant Filter reçoit les messages objets stockés dans la JMS Queue et effectue la recherche dans la zone ambiante limitée par le rayon R, pour sélectionner les composants de services qui correspondent au contrat de QoS exigé par l'utilisateur, et il récupère aussi l'adresse logique pour chaque composant de Service.