Rapport de stage de fin d'études DESS

-

Découverte dynamique de Web Services à travers la mise en œuvre des technologies du Web Sémantique

-

Centre de Recherche et Développement EDF (Clamart – 92)

Responsable de Stage : Philippe Bedu – chef du groupe TAIC

-

Hervé Perez

DESS Génie Des Logiciels Applicatifs – Université Paris VI - Jussieu

04-05-2004/30-09-2004

Sommaire

Résumé

Le Web Sémantique se veut une nouvelle infrastructure permettant d'accéder plus facilement et de manière automatisée aux ressources disponibles sur le web comme des documents et des services.
Pour cela, ces ressources devront faire l'objet d'une annotation sémantique afin de permettre une description formelle plus riche et précise de leurs contenus. Cette enrichissement sémantique ne pourra aller de pair qu'avec l'élaboration en parallèle de vastes ensembles d'ontologies qui regrouperont et organiseront l'ensemble des connaissances d'un domaine particulier.

L'annotation sémantique de web services est sans doute l'une des incarnation les plus importantes et les plus nécessaires à l'avènement du Web Sémantique. En effet, l'importance grandissante des web services dans les systèmes d'informations ainsi que dans les échanges inter-entreprises, entraîne la nécessité de disposer d'outils permettant de les manipuler aussi bien d'un point de vue sémantique que technique.

Le propos du stage est donc d'évaluer le degré de maturité des technologies mise à disposition par le W3C dans le cadre du Web Sémantique et de développer un prototype qui utilisera l'enrichissement sémantique de web services afin de réaliser une découverte dynamique de leur ordonnancement. Pour cela, le prototype mesurera la distance entre des concepts à l'aide d'inférences basées sur l'utilisation de relations de subsumptions.

Introduction

Les Web Services sont devenus ces dernières années prédominant dans la conception des architectures modulaires des systèmes d'informations. Cette prépondérance s'est avant tout affirmée grâce à l'utilisation de standards éprouvés tel que HTTP (Hypertext Transfer Protocol), ou bien encore, XML (Extensible Markup Language) qui permettent une interopérabilité accrue entre plateformes hétérogènes.

Aujourd'hui, les Web Services sont partout présent en nombres. Qu'il s'agisse de services de réservations en ligne ou bien de gestion de comptes bancaires et même d'applications métiers, tous ces services partagent en commun le fait d'être maintenant accessibles sous forme de Web Services.

Cette tendance ne semble pas devoir faiblir dans les prochaines années, bien au contraire. Ainsi, le monde du BtoB (Business to Business) et plus globalement celui du commerce électronique souhaitent disposer d'interactions encore plus flexibles et automatisées entre Web Services.

Cependant, cette volonté d'automatisation des interactions entre Web Services se heurte à un certain nombre de problèmes. En particulier, celui de posséder un mécanisme d'évaluation fiable et performant de l'interopérabilité effective d'un ensemble de Web Services.

Actuellement, la découverte dynamique de Web Services est gérée avec des annuaires qui en permettent la définition, le stockage et la découverte. L'un des deux principaux standard repose sur le protocole UDDI (Universal Description, Discovery and Integration) qui est un annuaire recensant exclusivement des Web Services et qui repose sur WSDL (Web Service Definition Language) pour leurs descriptions.

Il existe un autre standard aux visées plus étendues pour le commerce électronique nommé ebXML (Electronic Business eXtensible Markup Language) et qui recouvre l'ensemble des paradigmes proposé par UDDI et WSDL mais avec une méthode appelée UMM (UN Modeling Methodology) et qui repose sur UML.

Que ce soit UDDI ou ebXML, les algorithmes de découvertes des services utilisent une recherche avec des mots clés ou des tables de correspondances de couples (clés, valeurs) ce qui est très restrictif, trop sans doute. De plus, cette approche interdit de facto, la possibilité de combiner des Web Services.

Concernant l'interopérabilité entre Web Services, BPEL (Business Process Execution Language) s'affirme comme l'initiative la plus en vue pour permettre la détermination d'un ensemble de Web Services à exécuter. Malheureusement, la liste de Web Services est déterminée à l'avance, ce qui rend difficile la maintenance et l'évolutivité d'applications basées sur l'intégration de Web Services.

L'émergence de standards proposés par le W3C dans le cadre du Web Sémantique permet de proposer de nouvelles perspectives dans la résolution de ces problèmes.

Le projet

Problématique

La communauté spécialisée dans la conception d'architecture de système d'information s'accorde à considérer qu'il n'est plus raisonnable de tenter d'imaginer une cible universelle et immuable pour les systèmes d'information. Cela reviendrait à dire que les métiers, les interactions entre métiers, les concentrations, les organisations, les processus sont figés.

Or, l'ensemble des acteurs constatent que c'est de moins en moins le cas, et que la tendance est plutôt à une adaptabilité des systèmes à une ensemble de facteurs comme la fidélisation des clients, la fourniture de nouveaux services avec une qualité accrue, ou encore, une réactivité toujours plus importante.

Quel que soit le domaine d'activité, il devient évident que les systèmes d'informations devront permettre ce mode de fonctionnement, voire que la réorganisation continuelle des entités ne pourra pas se faire sans un système d'information idoine.

Aussi, il est important de trouver une voie permettant de comprendre fonctionnellement, quels seront les critères de collaboration entre entités et, comment chacune devra se présenter aux autres, déterminant ainsi comment traiter les choix de collaborations possibles.

Besoins

Tous les grands acteurs européen présent dans la production d'électricité travaillent à la conception de systèmes d'information où les technologies du Web Sémantique sont testées et évaluées afin d'élaborer les solutions du futur. EDF, acteur majeur du marché européen est impliqué de manière importante dans plusieurs projets où les techniques du Web Sémantique sont mises en oeuvre. Le groupe TAIC participe ainsi a l'évaluation des possibilités de l'exploitation de la connaissance utile à l'intégration d'applications collaborant à un processus métier paramétré.

L'objectif est donc d'utiliser les techniques d'ontologie (au sens défini par le W3C), de façon à réaliser une intermédiation de web services (web services matchmaking). Ces ontologies devront utiliser les standards proposés par le W3C et l'intermédiation devra être réalisée par un mécanisme d'inférences appelé raisonnement sur ces ontologies.
Dans un premier temps, EDF vise à évaluer l'effectivité d'une intégration sémantique d'applications dans le cadre du projet CIMERGY (Département Mire) et certainement à terme dans le cadre du défi réseau intelligent.

Le propos de mon stage est donc de déterminer les limites de cette approche, les avantages et les faiblesses et de mettre en œuvre dans un prototype l'ensemble de ces technologies.

Le prototype

Il a été décidé afin de mieux évaluer l'importance des changements que revêtirait l'adoption des technologies du Web Sémantique de se baser sur une application réelle reposant sur un ensemble de Web Services et servant de base à des recherches menées dans le cadre des échanges de RTE avec ses partenaires européens au sein d'ETSO.

RTE, le gestionnaire du Réseau de Transport d'Electricité pour la France est membre d'ETSO (European Transmission System Operators), une association regroupant trente-deux gestionnaires de réseau d'électricité provenant de quinze pays européens et qui sont répartis en cinq organisations régionales dont l'un des objectifs est l'harmonisation des accès aux réseaux électriques et à ses conditions d'usages.

Dans le cadre, de cette harmonisation, ETSO a proposé une normalisation des messages permettant à deux pays de planifier et valider une transaction portant sur une mise à disposition d'électricité entre deux pays. Cette normalisation se nomme ESS pour ETSO Scheduling System.

Un Ingénieur-Chercheur d'EDF, Cyril Effantin a travaillé sur la simulation d'une transaction ESS complète de mise à disposition d'électricité entre deux membres d'ETSO reposant sur un ensemble de Web Services.
Ces Web Services regroupés au sein d'un programme BPEL simulaient l'ensemble des phases nécessaire à la validation complète d'une transaction.

Il m'a donc été demandé de développer un prototype permettant d'effectuer la découverte dynamique d'un enchaînement de web services (workflow) en se basant sur les technologies proposées par le Web Sémantique.
Le workflow découvert devra représenter l'enchaînement séquentiel des web services nécessaires à l'exécution d'une transaction ESS. Cette découverte dynamique s'effectuera grâce à une intermédiation entre web services dont le résultat est obtenu après des inférences sur des ontologies représentant le domaine ayant pour cadre la normalisation des messages ETSO/ESS.

Le délivrable

Le délivrable à destination du groupe TAIC est composé des parties suivantes :

- l'ensemble des sources commentées de la plateforme.
- l'ensemble des sources commentées du prototype.
- la documentation programmeur du prototype au format javadoc.
- l'ensemble des sources des programmes annexes.
- le prototype compilé sous forme de fichier jar et le fichier bat permettant de l'exécuter.
- la présentation (fichier powerpoint) utilisée le 29/09/2004.
- le rapport de stage.

Le prototype a fait l'objet de plusieurs présentations à différents groupes et une étude détaillée de son fonctionnement à été faite en compagnie de Laurent Olivry (Ingénieur-Chercheur dans le groupe TAIC) au cours de deux séances de travail.

Introduction au Web Sémantique

Le Web Sémantique est une initiative du W3C, qui a officiellement vu le jour lors de la publication d'un article paru dans le numéro de mai 2001 du Scientific American et intitulé : « *The Semantic Web : A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities* ». On peut constater que le principal instigateur de cet article n'est autre que Tim Berners-Lee, l'inventeur du World Wide Web.

Le Web Sémantique vise à fournir un ensemble de marqueurs afin d'annoter le contenu des ressources présentes sur le web et ainsi rendre possible l'automatisation de nombreuses tâches encore aujourd'hui à la charge des utilisateurs.
Le Web Sémantique se veut donc d'abord une nouvelle infrastructure devant permettre à des agents logiciels d'aider plus efficacement différents types d'utilisateurs dans leur accès aux ressources disponibles sur le web (documents et services).

Pour cela, différents langages de niveau de complexités croissantes ont été proposé afin de mieux exploiter, combiner et raisonner sur les contenus de ces ressources. Ces nouveaux langages ont été définis afin de permettre une description formelle plus riche et précise des contenus.

Ces langages appartiennent à la famille des Logiques de Description. Ils permettent de disposer de différents niveaux d'expressivité et d'un nombre important de mécanismes d'inférences aussi appelés raisonnements.

Grâce à ces raisonnements, on peut aller au-delà des techniques syntaxiques de découverte basées sur des recherches à base de mots-clés et envisager d'utiliser des **critères de correspondance sémantique**. On pourra donc les rapprocher sur une base de points de correspondances inférés à partir de leurs définitions logiques.

Les connaissances permettant de s'affranchir des mots-clés seront utilisées sous forme de marqueurs sémantiques dans les ressources web et s'appuient sur des ontologies qui regroupent et organisent l'ensemble des connaissances d'un domaine particulier.

L'émergence d'ontologies communes à l'ensemble des acteurs du World Wide Web est l'un des défis majeurs à relever si on veut faire du Web Sémantique une réalité tangible.

Sémantique

Logique du premier ordre

L'approche symbolique en Intelligence Artificielle est une approche très répandue. La logique du premier ordre (logique de prédicat) est tout particulièrement favorisée, et découle d'une tradition philosophique que les spécialistes font remonter à Aristote.

Cette approche s'articule autour de deux axes :

- la représentation d'un domaine de connaissance par l'intermédiaire de symboles et au travers d'une classification (catégorisation, organisation) de ces symboles et la représentation des relations entre des objets particuliers ou des classes d'objets à l'aide de prédicats.

- le raisonnement qui autorise la combinaison de prédicats (opérateurs : et, ou, négation) pour permettre l'inférence de nouvelles connaissances (opérateur implication) et identifier l'équivalence de prédicats (opérateur équivalence)

La logique du premier ordre est donc capable de représenter un certain nombre de faits et de raisonner dessus mais la lecture de ces formules n'est pas aisée pour un être humain. Il faut donc créer des systèmes assez complexes pour permettre une interaction naturelle. Un certain nombre d'autres formalismes partiellement équivalents à la logique du premier ordre ont été développés comme les Logiques de Description.

Logiques de Description

Les recherches liées à la représentation de la connaissance et aux raisonnements ont conduit à des systèmes regroupés dans une catégorie nommée systèmes à base de connaissances.

Les Logiques de Description sont une des branches de l'étude de ces systèmes et sont issues des réseaux sémantiques et de frames. Les entités manipulées dans les Logiques de Description sont des concepts aussi appelés TBox, des rôles ou propriétés qui symbolisent les relations binaires entre objets et des individus représentant l'instance d'un de ces concepts (ABox).

Leurs définitions permet de les ordonner partiellement dans une hiérarchie appelée plus communément taxonomie et qui est basée sur une relation de généralité appelée **relation de subsomption**. Un concept A subsume un concept B si A est plus général que B.

Inférence

Dans sa définition classique, l'inférence est une opération logique portant sur des propositions tenues pour vraies (les prémisses) et concluant à la vérité d'une nouvelle proposition en vertu de sa liaison avec les premières.

Le système de raisonnement des Logiques de Description comprend deux mécanismes d'inférence principaux basés sur des calculs de relations de subsomption :

- la classification de concepts qui permet d'insérer un concept dans la hiérarchie.

- la reconnaissance d'instances donnant pour un individu les concepts les plus spécifiques dont il est instance.

D'autres raisonnements sur les concepts sont souvent utilisés comme :

- la satisfaisabilité d'un concept qui permet de savoir si un concept existe.
- l'équivalence entre deux concepts.
- la détermination du fait que deux concepts sont disjoints.

Ontologies

Le terme « Ontologie » à l'origine désigne une branche de la Philosophie qui s'occupe de l'organisation du Réel.

En informatique, les ontologies traitent de la représentation des connaissances d'un domaine particulier à travers l'utilisation de concepts et de rôles. L'intérêt de l'utilisation d'ontologies par des programmes et/ou des agents logiciels est qu'elle permet de partager la compréhension d'un domaine afin de pouvoir obtenir une interprétation sans équivoque ou ambiguïté de la signification des ressources présentent sur le web.

De RDF à OWL DL

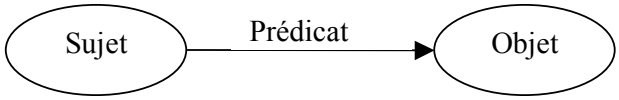
Pour que différentes applications ou agents logiciels puissent utiliser et échanger des métadonnées, il faut spécifier un modèle pour pouvoir les modéliser. RDF (Resource Description Framework) recommandé par W3C (depuis le 10/02/2004) est aujourd'hui utilisé comme un standard pour cet objectif.

Les métadonnées sont décrites sous la forme d'un triplet :

- Sujet : La ressource que l'on définie.
- Prédicat : La propriété de la ressource.
- Objet : Valeur de la propriété pouvant être une autre ressource ou bien un littéral.

Ces métadonnées permettent de décrire des ressources web ou des relations entre ces ressources. RDF, est à la base exprimé sous la forme d'un graphe orienté mais on peut le sérialiser sous différentes syntaxes dont XML (On parle alors de "RDF/XML"). D'autres sérialisations existent comme N-Triples.

Figure 1 – Exemple de représentation d'un triplet sous forme de graphe.



Le même triplet sérialisé au format RDF/XML :

<rdf:Description about=***sujet***>
    <***prédicat***>***objet***</***prédicat***>
<rdf:Description/>

Si RDF fournit une capacité d'échange de description de ressources, il ne permet pas à l'utilisateur de définir un vocabulaire de termes et d'établir des relations entre ces termes qui portent une sémantique de description de ces ressources.
Un schéma RDF avec son apport peut offrir cette capacité. Le schéma est lui-même défini sous RDF et on l'appelle RDFS. RDFS fournit la notion par exemple de classe (rdfs :Class), de sous-classes (rdfs :subClassOf) ou bien encore de sous-propriétés (rdfs :subPropertyOf).

Malgré les apports de RDFS à RDF, des manques apparaissent quand il s'agit d'élaborer des ontologies. Ainsi, il n'y a pas de distinctions entre les classes et les instances (individus) et on ne dispose pas de la possibilité d'indiquer des contraintes sur un domaine, pas plus que sur les cardinalités ou bien encore de préciser qu'une propriété est transitive, inverse ou symétrique.

L'ensemble de ces manques fait de RDF Schéma un support insuffisant pour répondre aux exigences du Web Sémantique.

Des propositions de nouveaux langages ont été faites avec comme cahier des charges d'étendre les standards existants, à savoir RDF/RDFS et se devaient de permettre un pouvoir d'expression suffisant ainsi que le support du raisonnement automatisé.

Deux langages ont été développés pour répondre à ces besoins, OIL (Ontology Inference Layer) majoritairement européen et DAML (DARPA Agent Markup Language) majoritairement Américain.
Des efforts ont été faits pour produire DAML + OIL et donc d'étendre le sous-ensemble des Logiques de Description de RDF. DAML + OIL a été soumis au W3C comme une base pour une standardisation. Dans la foulée, le Web-Ontology Working Group a été créé et OWL (Ontology Web Language) est né du travail fait à partir de DAML+OIL.

OWL est maintenant une recommandation du W3C (depuis le 10/02/2004) et se décline en trois versions :

- OWL Lite qui définit une hiérarchie de classification et fournit la possibilité d'ajouter des contraintes simples.

- OWL DL qui intègre les Logiques de Descriptions. Cela permet une expressivité maximum sans pour autant perdre la complétude du calcul (toutes les inférences sont assurées d'être prises en compte) et la décidabilité (tous les calculs seront terminés dans un intervalle de temps fini) des systèmes de raisonnement.

  OWL DL inclut toutes les structures de langage de OWL, avec des restrictions comme la séparation des types (une classe ne peut pas aussi être un individu ou une propriété et une propriété aussi être un individu ou une classe).

  La maturité d'OWL DL est assurée par les nombreuses années de recherches dans le champ des Logiques de Descriptions.

- OWL Full est une union de la syntaxe OWL et de RDF permettant une expressivité maximale et la liberté syntaxique de RDF mais sans garantie de décidabilité des calculs dans un temps fini. Ainsi, une classe peut se traiter simultanément comme une collection d'individus et comme un individu à part entière. Une autre différence significative par rapport à OWL DL réside dans la possibilité que possède une ontologie OWL full à pouvoir augmenter la signification du vocabulaire prédéfini (RDF ou OWL).

Actuellement, aucun système de raisonnement ne met en œuvre toutes les caractéristiques de OWL Full.

La compatibilité entre ces trois catégories est assurée dans le sens ascendant uniquement.

Le langage OWL est une des contribution majeure pour la mise en œuvre du Web Sémantique. Sa capacité, en autre à pouvoir incorporer d'autres ontologies à travers un système « d'import » est fondamental pour le développement rapide de tout un ensemble d'ontologies. Cette approche, voisine de la conception objet permettra à tout en chacun de bénéficier d'ontologies de domaines répondant à ses besoins. Dans le cas échéant, le développement d'une ontologie devrait être minimal en regard de la disponibilité d'ontologies déjà existantes et couvrant déjà partiellement le domaine concerné.

En outre, le langage OWL présuppose un *monde ouvert, c*'est-à-dire que les descriptions de ressources ne se confinent pas à un seul fichier où à une seule portée.

Conception et développement des ontologies ETSO

Le développement d'ontologie était l'une des parties les plus importante du projet. En effet, de la bonne conception des ontologies dépendait la possibilité d'effectuer des inférences permettant de déterminer les relations entre les concepts liés à chacun des Web Services susceptibles d'intervenir dans le processus de découverte dynamique d'un ordonnancement de Web Services.

Cependant, la seule détermination de la relation entre concepts n'est pas suffisante pour déterminer si deux Web Services sont compatibles. En effet, dans un processus de découverte dynamique d'un ordonnancement de Web Services, la seule affirmation que les concepts soient les mêmes, équivalents, ou bien ayant une relation de subsomption n'est pas déterminante.

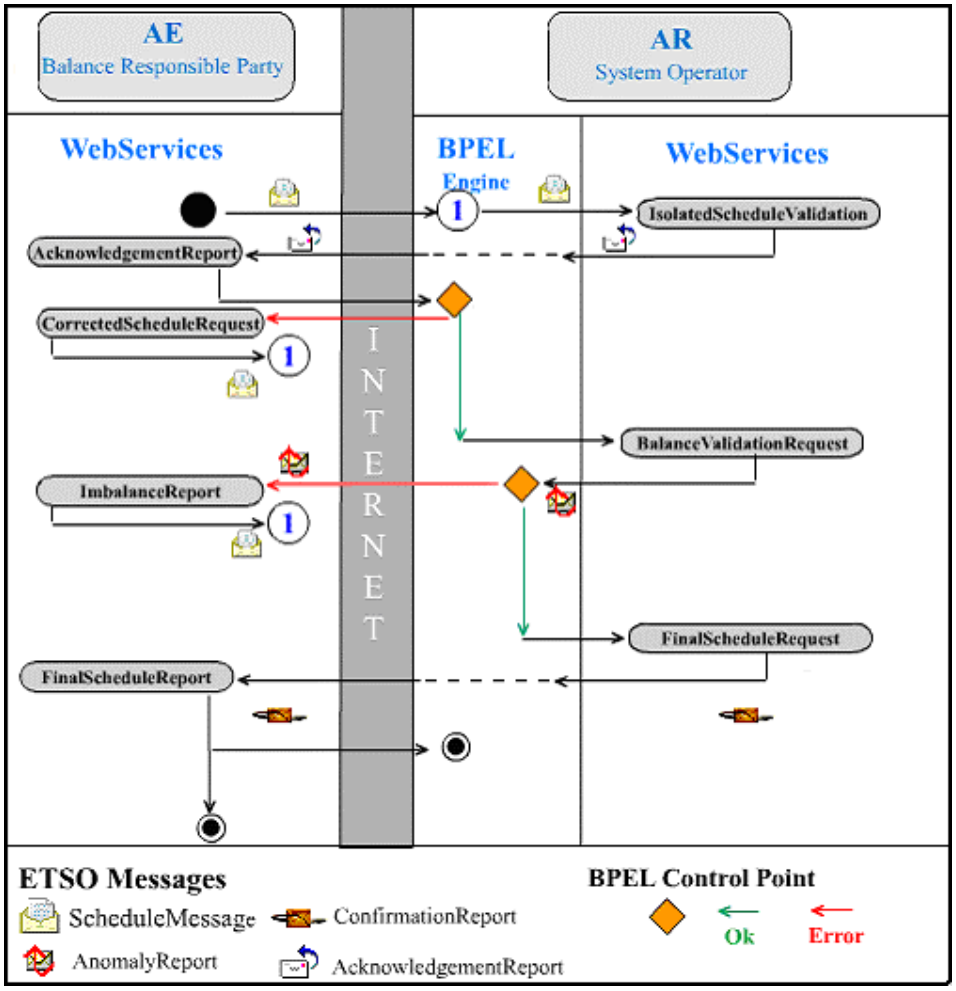
On doit donc pour déterminer cet ordonnancement, tenir compte des paramètres en entrée et en sortie afin de pouvoir réellement disposer d'un moyen de découvrir cet ordonnancement. Pour cela, une ontologie relative aux paramètres a été développée en plus de celle regroupant l'ensemble des concepts ETSO afin de pouvoir effectuer des inférences sur les paramètres qui seront considérés au sein de cette ontologie comme des concepts.

A partir de ces deux inférences successives, on pourra déterminer un ordonnancement de Web Services en validant la pertinence du web service grâce au concept qui le définit dans l'ontologie ETSO et vérifiant l'équivalence des concepts entre les paramètres d'entrée et de sortie de deux web services.

Ontologie des Concepts ETSO

A partir du schéma définissant une transaction ETSO à travers un ensemble de Web Services (voir figure 2) fournit par Cyril Effantin, ma première tâche à été de concevoir une ontologie qui permettrait de recenser et d'organiser en concepts cet ensemble de Web services.

Figure 2 - Liste des Web Services intervenant dans la validation ESS.



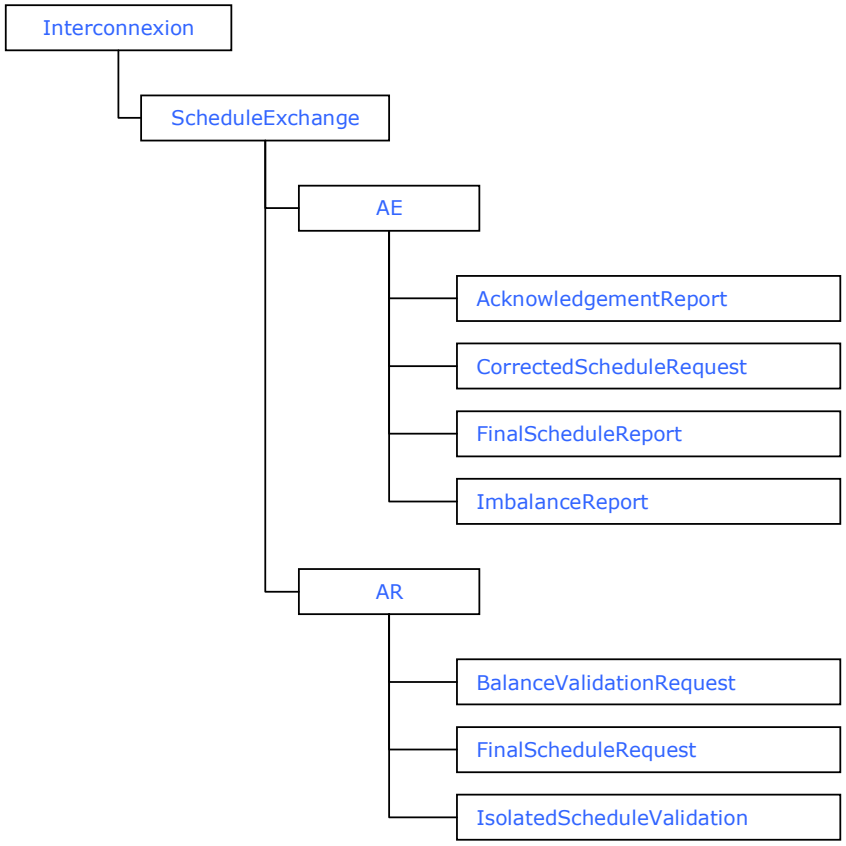
La nature même d'une transaction fait immédiatement apparaître deux catégories de Web Services. Cette organisation se retrouvera dans l'ontologie des concepts où les Web Services appartiennent à une des deux catégories suivant qu'ils sont mis en œuvre par l'émetteur, ici incarné par le *Balance Responsible Party (AE)* ou bien par le récepteur *System Operator (AR)*.

L'ontologie des concepts a pour utilité première de vérifier que les Web Services qui feront l'objet d'une intermédiation valideront une relation minimun de subsomption avec le concept associé à une transaction de type ETSO. Pour cela, la catégorie *ScheduleExchange* regroupant l'ensemble des concepts ayant trait à un échange planifié entre deux entités a été créée. La convention de nommage adoptée pour désigner les concepts reprend le nom des web services.

Même si cette ontologie minimaliste est suffisante dans le cadre du prototype, je me suis rendu sur le site d'ETSO (http://www.etso-net.org) afin de recenser le plus de concepts possibles associés à l'activité de cette organisation pour disposer d'une ontologie aussi exhaustive que possible. Il faut cependant préciser que le développement d'une ontologie complète d'un domaine est un

travail qui aurait pu très facilement faire l'objet d'un stage à part entière et aurait demander d'être en contact avec des représentants de cette organisation.

Figure 3 – Extrait de l'ontologie des Concepts ETSO montrant l'organisation des concepts liés aux échanges de messages planifiés.



Extrait de l'ontologie OWL des Concepts ETSO correspondant au schéma ci-dessus.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://etso.com/concepts#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://etso.com/concepts">

   <owl:Ontology rdf:about=""/>

    <owl:Class rdf:ID="Interconnexion">
      <rdfs:subClassOf rdf:resource="#Etso"/>
    </owl:Class>
    <owl:Class rdf:ID="ScheduleExchange">
       <rdfs:subClassOf>
           <owl:Class rdf:about="#Interconnexion"/>
       </rdfs:subClassOf>
```

```
    </owl:Class>

    <owl:Class rdf:ID="AE">
       <rdfs:subClassOf rdf:resource="#ScheduleExchange"/>
    </owl:Class>

    <owl:Class rdf:ID="AR"></owl:Class>

    <owl:Class rdf:ID="FinalScheduleRequest">
       <rdfs:subClassOf rdf:resource="#AR"/>
    </owl:Class>

    <owl:Class rdf:ID="BalanceValidationRequest">
       <rdfs:subClassOf rdf:resource="#AR"/>
    </owl:Class>

    <owl:Class rdf:ID="CorrectedScheduleRequest">
       <rdfs:subClassOf>
           <owl:Class rdf:about="#AE"/>
       </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:ID="ImbalanceReport">
       <rdfs:subClassOf>
           <owl:Class rdf:about="#AE"/>
       </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:ID="FinalScheduleReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#AE"/>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="AcknowledgementReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#AE"/>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="IsolatedScheduleValidation">
    <rdfs:subClassOf rdf:resource="#AR"/>
  </owl:Class>
</rdf:RDF>
```

Ontologie des Paramètres

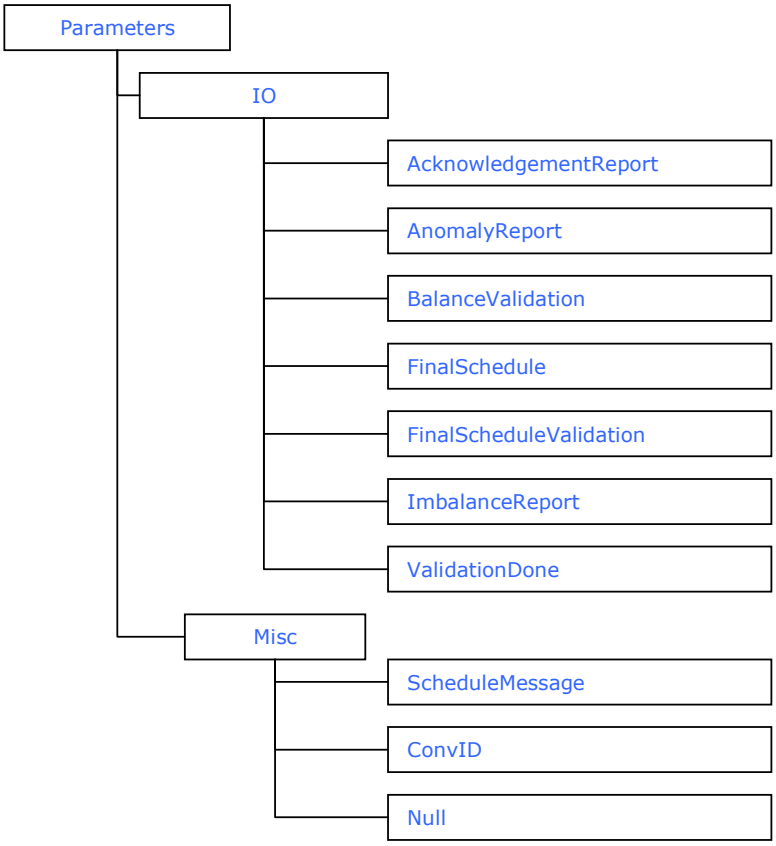
La nécessité de posséder une ontologie spécifique aux paramètres s'est imposée une fois que l'algorithme d'intermédiation des Web Services a été déterminé.

Elle aurait très bien pu être intégrée à l'ontologie des concepts ETSO mais sa nature spécifique liée à la catégorisation des paramètres d'entrée et de sortie des web services la rendait indépendante du domaine ETSO. On peut justifier cette

« externalisation » ne serais-ce qu'en terme d'évolutivité. En effet, une ontologie spécifique sera plus appropriée en cas d'évolution de la signature d'un web service.

Dans cette ontologie, les concepts sont l'expression de l'ensemble des paramètres en entrée et sortie. Ils sont regroupés ensemble car un paramètre en sortie sera au niveau du web service suivant présent sous la forme d'un paramètre en entrée. Cette dualité indispensable pour pouvoir effectuer la découverte dynamique fait que l'ensemble des paramètres sont regroupés dans une catégorie nommée IO (InputOutput).
Ici, encore la convention de nommage adoptée pour désigner les concepts reprend le nom des paramètres.

Figure 4 – Extrait de l'ontologie des Paramètres des web services ETSO.

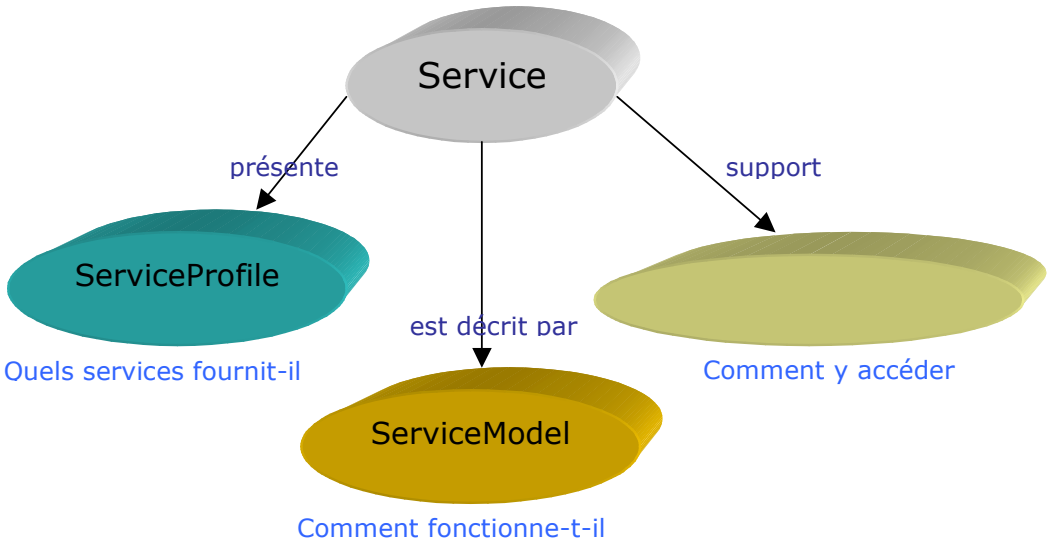


OWL-S

OWL-S (Ontology Web Language for Service) est une ontologie basée sur OWL et spécifiquement conçue pour les Web Services. OWL-S fournit un ensemble de marqueurs afin de décrire les propriétés et les capacités de Web Services. OWL-S a été conçu pour faciliter l'automatisation des tâches relatives aux Web Services et, plus particulièrement la découverte automatique de Web Services, leurs exécutions, ou bien encore leurs compositions et leurs interopérabilités.

Les versions précédentes d'OWL-S étaient connues sous le nom de DAML-S et la version actuelle est la 1.0 en date de novembre 2003. La béta-version 1.1 est disponible depuis juillet 2004.

Figure 5 - Architecture OWL-S.



OWL-S est architecturé autour de quatre entités dont la classe Service fournit le point d'entrée pour la description d'un web service. Il n'existe qu'une seule instance de la classe Service par Web Service.

Le *ServiceProfile* renseigne sur les services proposés par le Web Service. Il permet à des agents logiciels d'accéder à l'ensemble des informations nécessaires leur permettant d'évaluer si le Web Service correspond à leurs besoins.

Le *ServiceModel* permet de connaître le fonctionnement du service. Il peut être utilisé de quatre manières différentes :

- Permet d'effectuer une analyse plus complète du Web Service.
- Permet de composer des descriptions de service à partir de multiples services pour réaliser une tâche spécifique
- Permet de coordonner les activités des différents participants à l'exécution de ce service.
- Permet de surveiller l'exécution du service.

Le *ServiceGrounding* définit les modalités d'accès au service. Typiquement, le *ServiceGrounding* va spécifier un protocole de communication, les formats des messages ainsi que les autres détails spécifiques ayant trait à la mise en œuvre technique du service.

De plus, le *ServiceGrounding* sert à définir pour chaque type abstrait, une manière claire d'échanger des données de ce type. A l'instar de l'ensemble

des fichiers OWL-S développés pour le prototype utilise pour la définition des paramètres entrée/sortie, les concepts de l'ontologie des Paramètres.

Au niveau des contraintes sur les cardinalités, OWL-S n'en spécifie que deux. Un service doit être décrit au plus par un *ServiceModel* et doit posséder un seul *ServiceGrounding*.

Comme le programme de découverte dynamique d'un Workflow de Web Services va se baser sur une intermédiation, le *ServiceProfile* va requérir une attention toute particulière. En effet, le *ServiceProfile* nous permet d'accéder à l'ensemble des informations nécessaires pour l'algorithme d'intermédiation.

Le ServiceProfile

Le *ServiceProfile* fournit plusieurs types d'informations indispensable au bon fonctionnement de l'algorithme d'intermédiation.

Tout d'abord, on retrouve des informations sur l'entité qui fournit ce Web Service. Cette information dans le prototype est présente au niveau du web service répertorié sur le serveur UDDI.

Le ServiceProfile fournit aussi une description fonctionnelle en termes de paramètres d'entrées requis pour l'exécution du service et de paramètres de sortie générés par le service. On retrouve en outre, des les préconditions et les effets.
On nomme ces quatres éléments (paramètres inclus) : IOPE (input, output, préconditions, effect). Les préconditions symbolisent les conditions nécessaires à la bonne exécution du service.

Dans le cadre du prototype, aucune Préconditions ou Effets n'est implémenté. Cependant, l'expression de contraintes préalable à la sélection d'un Web Service devrait en principe être formulé à cet endroit avec par exemple un langage de règles comme SWRL (Semantic Web Rule Language).

Le ServiceProfile fournit d'autres propriétés qui sont utilisées pour décrire les fonctionnalités du service notamment le ServiceCategory qui sert à classifier le Web Service en rapport à une ontologie ou une taxonomie. Cette propriété est très proche de la propriété homonyme de UDDI. J'ai utilisé cette propriété dans l'ensemble des fichiers OWL-S afin d'indiquer le concept définissant le service. Il correspond exactement au serviceCategory renseigné dans l'annuaire UDDI.

Le serviceParameter lui définit une liste de propriétés qui peuvent être utilisé par le serviceProfile.
Le QualityRating permet d'apprécier et d'évaluer la qualité d'un service. Dans une optique de e-business, ce genre d'information pourrait se révéler très intéressante et pourquoi pas à la charge d'organismes indépendants chargés d'évaluer en toute objectivité la qualité d'un service.

Dans le cadre du processus d'intermédiation de Web Services, seul le ServiceProfile est réellement utilisé car les informations proposées sont suffisantes pour effectuer l'ensemble des opérations nécessaires.

Réalisation des fichiers OWL-S

L'ensemble des Web Services participant à la transaction ETSO se sont vu attribués un fichier OWL-S. Pour des raisons de simplicité l'accès à l'ensemble des informations, le Profile, le Process et le Grounding sont localisés avec des URIs « pointant » dans le même fichier que le Service.

La réalisation de ces fichiers s'est effectuée en deux phases. En premier lieu, j'ai utilisé un outil proposé par mindswap, le Maryland Information and Network Dynamics Lab Semantic Web Agents Project (http://www.mindswap.org) et nommé WSDL2OWL qui à partir du WSDL de chaque Web Service génère un fichier OWL-S correspondant aux données contenues dans le WSDL. Cette génération automatique n'apporte qu'environ 50% des données nécessaires à une utilisation effective dans le cadre du prototype.

En effet, si le process et le grounding sont relativement bien renseignés, le profile lui est à définir complètement. Et, comme expliciter précédemment cette partie est la plus importante pour garantir le bon fonctionnement de l'algorithme d'intermédiation. Les informations de nature sémantiques ne sont malheureusement pas facilement déductibles à partir du WSDL.

Des initiatives apparaissent pour essayer d'automatiser le plus possible cette phase « d'enrichissement sémantique ». La plus intéressante est sans doute celle de Assam Annotator (Automated Semantic Service Annotation With Machine Learning) de l'university College de Dublin.
Assam est un outil intéressant mais seulement dans une optique de long terme. En effet, ce sont des résultats de l'apprentissage que dépend la bonne annotation des fichiers OWL-S. Le faible volume de Web Services (7) dont je disposai n'aurait sans doute pas permis un apprentissage suffisant pour obtenir de bon résultat sans compter que l'outil est encore en plein développement.

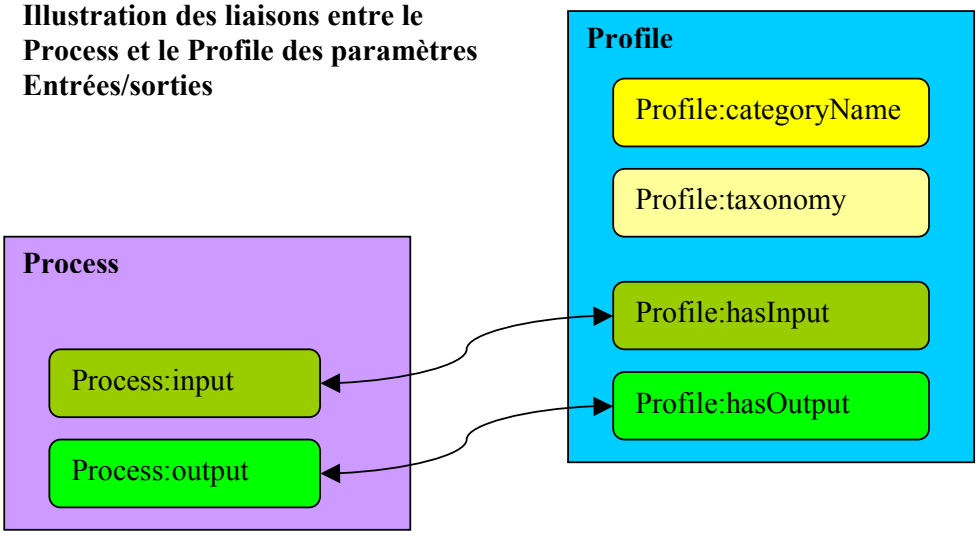
Compléter le Profile de l'OWL-S a donc consisté à renseigner le ServiceCategory, c'est-à-dire d'indiquer à quel concept est rattaché le Web Service. Le concept appartenant à la catégorie *ScheduleExchange* de l'ontologie *ETSO .* La première inférence sera effectuée à partir de l'extraction du ServiceCategory de chaque Web service.

Le ServiceCategory contient les propriétés suivantes :

- Code : renseigne un code quelconque mais afin de garder un maximum de similarités entre l'enregistrement des web services sur le serveur UDDI et le fichier OWL-S, j'ai repris le même code issue du NAICS (voir partie UDDI)
- Value : renseigne le libellé du code.
- Taxonomy : la propriété la plus importante car elle permet de renseigner l'URI de l'ontologie qui est associé au CategoryName.
- CategoryName : renseigne le concept catégorisant le Web Service.

L'autre partie dans le ServiceProfile à renseigner concerne les paramètres en entrée et sortie qui même si OWL-S 1.0 autorise l'inconsistance entre le profile et le Process doivent être similaires. La déclaration des paramètres en entrée comme en sortie se fait à l'aide d'URIs pointant sur les paramètres déclarés dans la classe Process.

Illustration des liaisons entre le Process et le Profile des paramètres Entrées/sorties



Profile OWL-S du Web Service AcknowlegmentReport.

```
<profile:Profile rdf:ID="AcknowledgementReportProfile">
   <service:isPresentedBy rdf:resource="#AcknowledgementReportService"/>

   <profile:serviceName xml:lang="en">
         AcknowledgementReport
   </profile:serviceName>

   <profile:textDescription xml:lang="en"></profile:textDescription>

   <profile:hasInput rdf:resource="#AcknowledgementReport"/>
   <profile:hasInput rdf:resource="#ConvID"/>
   <profile:hasOutput rdf:resource="#BalanceValidation"/>

   <!-- Service Category -->
   <profile:ServiceCategory rdf:ID="SC">

    <profile:code rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    22112
    </profile:code>

    <profile:value rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Electric power transmission, control, and distribution
    </profile:value>

   <profile:taxonomy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   http://127.0.0.1:8080/edf.web.semantique/files/ontologies/ConceptsETSO.owl
   </profile:taxonomy>

   <profile:categoryName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   ScheduleExchange
```

```
        </profile:categoryName>

        </profile:serviceCategory>
    </profile:Profile>
```

Les déclarations des paramètres dans le Process, se fait au moyen de l'utilisation de ressources (au sens RDF) et non de littéraux sur l'ontologie des paramètres. Cela permet de référencer les paramètres comme des concepts localisés dans une ontologie et donc de bénéficier d'une conception de fichier OWL-S reposant entièrement sur une définition à base d'ontologies. En revanche, cela impliquera par contre, si l'on désire exécuter le web service en interprétant directement le grounding d'inclure par exemple une transformation xsl au sein par d'une balise cdata afin de déclarer par exemple que le paramètre « ConvID » est une chaine de caractères ou un entier.

Extrait du Process illustrant la déclaration des trois paramètres du Web Service AcknowlegmentReport.

```
<!-- Paramètre AcknowledgementReport -->
<process:Input rdf:ID="AcknowledgementReport">
  <process:parameterType
      rdf:resource="http://127.0.0.1:8080/edf.web.semantique/files/ontologies/ParamsETSO.owl#AcknowledgementReport">
  </process:parameterType>
   <rdfs:label>AcknowledgementReport</rdfs:label>
</process:Input>

<!-- Paramètre ConvID -->
<process:Input rdf:ID="ConvID">
  <process:parameterType
      rdf:resource="http://127.0.0.1:8080/edf.web.semantique/files/ontologies/ParamsETSO.owl#ConvID">
  </process:parameterType>
  <rdfs:label>ConvID</rdfs:label>
</process:Input>

<!-- Paramètre BalanceValidation -->
<process:Output rdf:ID="BalanceValidation">
  <process:parameterType
      rdf:resource="http://127.0.0.1:8080/edf.web.semantique/files/ontologies/ParamsETSO.owl#BalanceValidation">
  </process:parameterType>
  <rdfs:label>BalanceValidation</rdfs:label>
</process:Output>
```

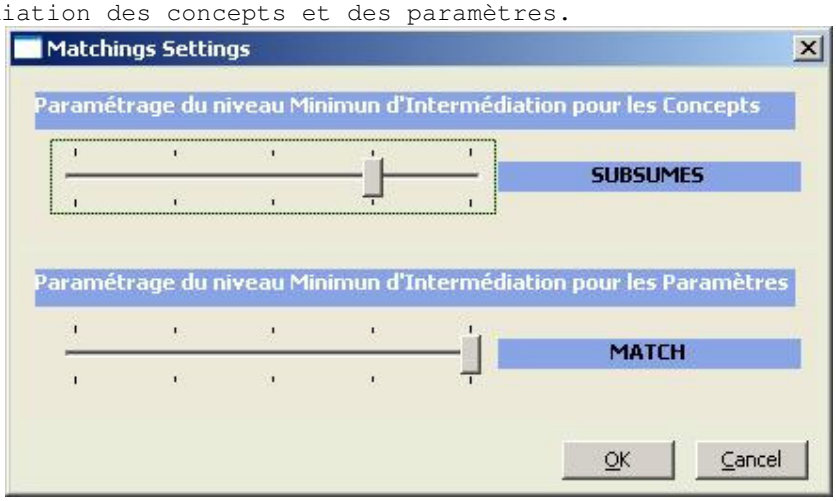
Intermédiation des Web Services

L'intermédiation des web services, où comment mesurer le degré de « comptabilité ou similarité » entre deux web services est le cœur de la problématique du stage. Cette intermédiation se définit plus précisément comme la mesure d'un indice de satisfaction évalué grâce à la distance des concepts mesurée à l'aide d'inférences.

En effet, en parvenant à déterminer cet indice de satisfaction entre deux web services on pourra déterminer l'ordonnancement effectif d'un ensemble de Web Services.

Ce degré de « comptabilité » peut varier en fonction des besoins de l'utilisateur (agents logiciels autonomes ou humains) et dans le prototype (voir figure 6) on dispose de la possibilité de régler précisément l'indice de satisfaction minimum que l'on considère comme indispensable à une intermédiation réussie.

Figure 6 – Copie d'écran de la fenêtre permettant de régler le niveau d'intermédiation des concepts et des paramètres.



Cet indice de satisfaction où score après une inférence entre les deux concepts (A et B) peut revêtir les valeurs suivantes :

- Equivalence (match)
  A et B sont considérés comme équivalents et ils ne représentent qu'un seul et même concept.
- Subsomption (subsumes) :
  Le concept B est subsumé par le concept A ce qui signifie que le concept A est plus général que B.
- Subsomption Inverse (invert subsumes):
  Le concept A est subsumé par le concept B ce qui signifie que le concept B est plus général que A.
- Non Classification (unclassified) :
  Le concept B n'est lié à aucun concept de l'ontologie référençant le concept A.
- Echec (fail) :
  Aucune des valeurs précédentes n'a été retenue.

Algorithme d'intermédiation

L'intermédiation portera à la fois sur les concepts représentant les web services dans une ontologie et sur les concepts associés aux paramètres des différents web services. En effet, déterminer qu'un web service appartient à un

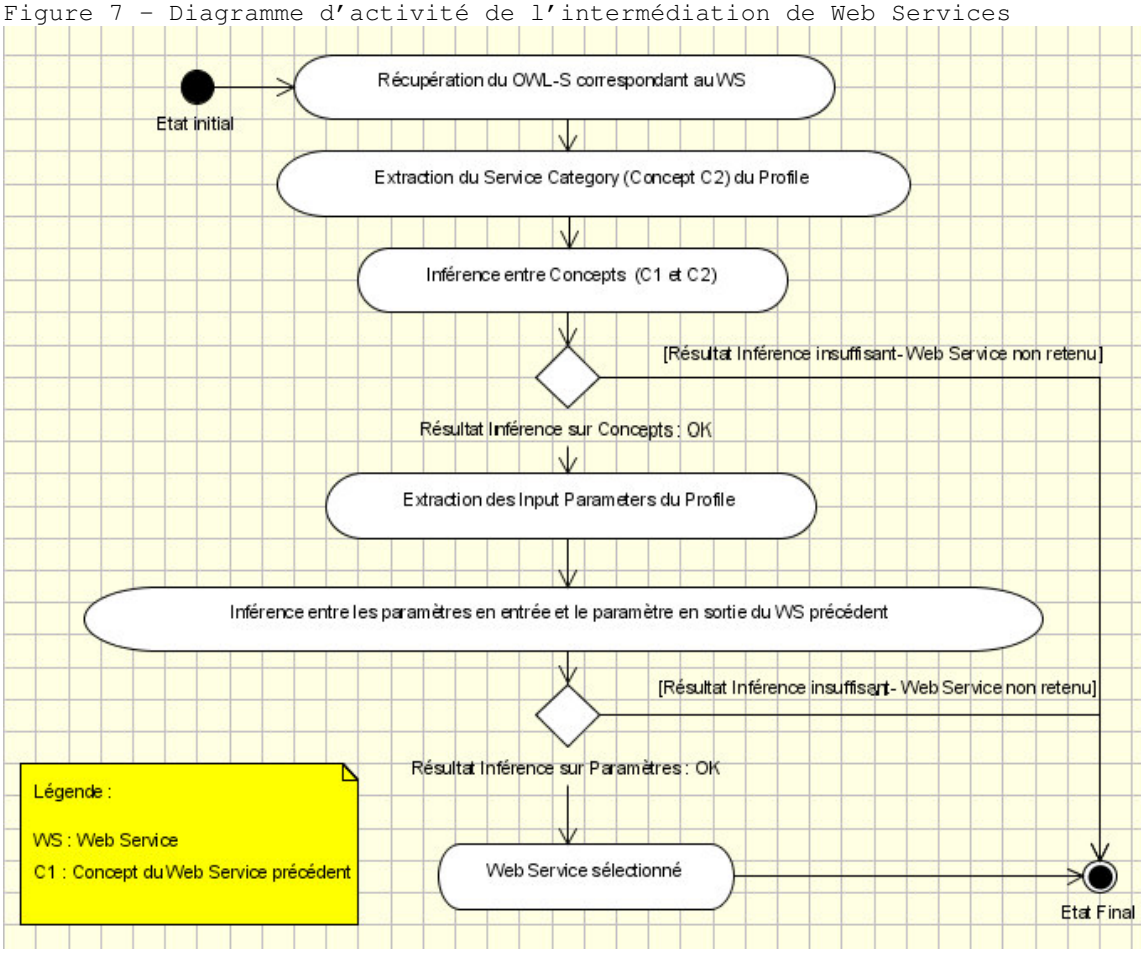
domaine proche de celui recherché n'est pas suffisant pour pouvoir déterminer l'ordonnancement hiérarchique d'un ensemble de web services.

L'algorithme d'intermédiation entre deux web services s'effectue en deux passes tel que décrit dans le diagramme d'activité (voir figure 7) :

- La première passe vise à déterminer si les deux web services appartiennent au même domaine. Pour cela, on déterminera le score résultant de l'inférence entre les concepts représentant chacun le domaine des web services. Cette première passe permet de valider le fait que les deux web services appartiennent au même domaine et que par conséquent la présence du web service testé dans le workflow est justifiée. Les concepts sont extraits des fichiers OWL-S associés à chacun des web services, plus précisément, on récupère le « CatégoryName » dans le « ServiceCategory » de la partie Profile.

  Si le score obtenu après la première passe est insuffisant, le web service sera rejeté car son domaine ne sera pas jugé suffisamment « proche » du précédent web service. En cas de score suffisant, l'algorithme d'intermédiation passe à la phase 2.

Figure 7 – Diagramme d'activité de l'intermédiation de Web Services

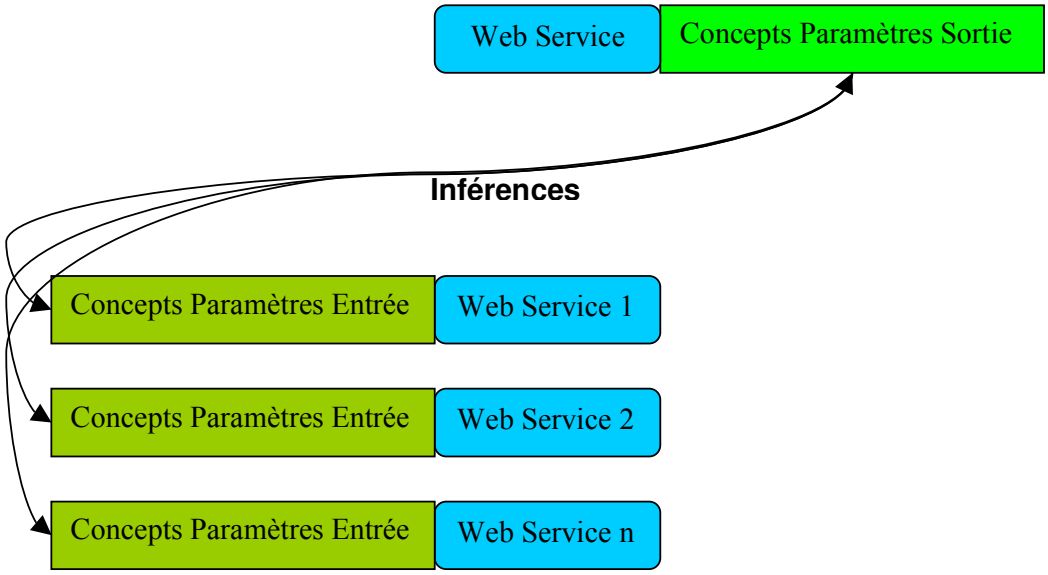


- La deuxième phase, elle s'applique à déterminer si le concept associé au paramètre en sortie du web service précédent sera après inférence proche ou équivalent d'un des paramètres en entrée du web service testé. L'inférence liée aux tests de subsumption sera déterminante dans l'évaluation du workflow. En effet, seule la détermination d'une distance suffisante entre les concepts des paramètres entrée/sortie peut permettre de découvrir un ordonnancement logique d'un ensemble de web services.

Si le web service testé est considéré comme à la fois proche ou équivalent du domaine du web service précédent et dont l'un des paramètres en entrée possède un concept proche ou équivalent (suivant le paramétrage voulu) du concept associé au paramètre en sortie du web service précédent alors le web service est sélectionné.
Il deviendra à son tour le web service de référence dont le concept et le paramètre de sortie serviront de référence pour la poursuite de la découverte dynamique du workflow.

Figure 8 – Schéma montrant la mise en relations des paramètres entrée/sortie lors de l'inférence.

Inférences entre les concepts paramètres sortie et paramètres entrée



Pour être plus précis, il est possible que plusieurs web services correspondent à l'ensemble des critères exigés et soient donc sélectionnés à chaque étape du workflow. Pour gérer cela, un mécanisme de pile stocke l'ensemble des web services découvert à chaque étape ce qui permet de couvrir l'ensemble des possibilités du workflow.

Chaque étape donne lieu à l'attribution d'une note de 0 à 4 en fonction du score obtenu lors de l'inférence.

- L'équivalence attribue un 4.
- La subsomption attribue un 3.

- La subsomption Inverse attribue un 2.
- La non classification attribue un 1
- Un échec attribue un 0.

Un score final de l'intermédiation entre deux web services est calculé et sauvegardé. Actuellement, il s'agit d'une simple addition des deux notes mais une méthode de calcul plus complexe pourrait être mise en œuvre dans le cadre d'une évolution du prototype afin de pondérer différemment l'inférence sur le concept associé au web service et l'inférence sur les concepts associés aux paramètres.

Ce score final pourrait ainsi servir à déterminer le parcours de workflow le plus pertinent lorsque que plusieurs web services ont été découvert à une étape du workflow.

En outre, un mécanisme permet de vérifier si le nouveau web service retenu ne pointe pas sur un des web services déjà retenu précédemment dans le workflow. Cela évite de se retrouver dans la configuration d'un graphe cyclique qui entraînerait l'algorithme à boucler indéfiniment.

Evaluation des besoins

Matériel de base

Cyril Effantin m'a fournit un ensemble de web services, plus précisément l'ensemble des fichiers WSDL correspondant à chacun des web services intervenant dans la simulation de la transaction ESS. Il m'a aussi fournit une animation flash montrant le déroulement de l'ensemble de la transaction (voir figure 2) afin que je puisse me faire une idée de l'ordonnancement optimal des web services.

Cette notion d'ordonnancement est importante car elle interviendra lors de la conception des ontologies qui joue un rôle de premier plan dans la possibilité fournie au démonstrateur de découvrir un ensemble de Web Services pouvant s'enchaîner.

Analyse des besoins fonctionnels

La mise en œuvre d'un prototype répondant aux attentes du cahier des charges a nécessité l'étude d'un ensemble de composants logiciels pouvant intervenir dans la mise en œuvre du prototype. La phase d'analyse s'est déroulée au début de la deuxième partie du stage. La première partie du stage ayant été consacrée à l'étude de l'état de l'art des technologies proposées par le W3C dans le cadre du Web Sémantique et à leurs évaluations dans le cadre d'une mise en œuvre pratique à travers le prototype.

Après l'étude de différentes configurations possible, il est apparu que les éléments suivant devaient être proposés et associés afin de parvenir à une architecture logicielle pouvant répondre aux exigences du cahier des charges.

- **Plateforme d'hébergement des Web Services et des fichiers connexes (ontologies, fichiers OWL-S)**

  L'hébergement des web services ETSO demandait de posséder une plateforme capable de les accueillir et de pouvoir les exécuter. La nécessité, en outre, de devoir accéder à un contenu de nature sémantique par définition étroitement lié aux web services plaidait pour une plateforme aux rôles multiples.

  Ainsi, en plus des web services, la plateforme devait pouvoir héberger les ontologies du domaine ETSO au nombre de deux et les fichiers OWL-S qui sont associés à chacun des Web Services.

  La plateforme devait permettre une consultation aisée des données disponibles sur la plateforme que ce soit la liste des web services ou bien encore le contenu d'un fichier OWL-S associé à un web service.

- **Serveur UDDI**

  Très vite, le fait de disposer d'un serveur UDDI (Universal Description, Discovery and Integration) s'est imposé. En effet, le besoin de posséder un annuaire référençant un ensemble de Web Services a été nécessaire. Dans une situation réelle, un programme d'intermédiation de Web services doit disposer d'un moyen d'accéder à un vaste catalogue de Web Services.
  UDDI, outre sa capacité à recenser une liste de web services possède l'immense avantage de pouvoir les organiser et les catégoriser.
  Sans cette capacité, la liste potentiellement gigantesque du nombre de web services pouvant être hébergés sur un serveur UDDI rendrait caduque toute velléité de découverte d'un workflow de web services dans un temps de calcul raisonnable.
  Le protocole UDDI permet donc d'effectuer des recherches précises sur la base de différents critères.
  Le serveur UDDI sert donc à la fois d'annuaire et de filtre pour obtenir un ensemble de web services qui seront traités par le programme de découverte dynamique du workflow.

- **Moteur d'inférences**

  La découverte du workflow de web services se base très précisément sur la capacité à pouvoir effectuer des inférences sur des concepts liés au domaine ou catégorie associé à un web service ainsi qu'à ses paramètres. Pour cela, il fallait pouvoir disposer d'un moteur d'inférences pouvant déterminer et mesurer la distance entre les différents concepts.
  La principale caractéristique de ce moteur devait être qu'il puisse travailler nativement avec des ontologies de type OWL DL.

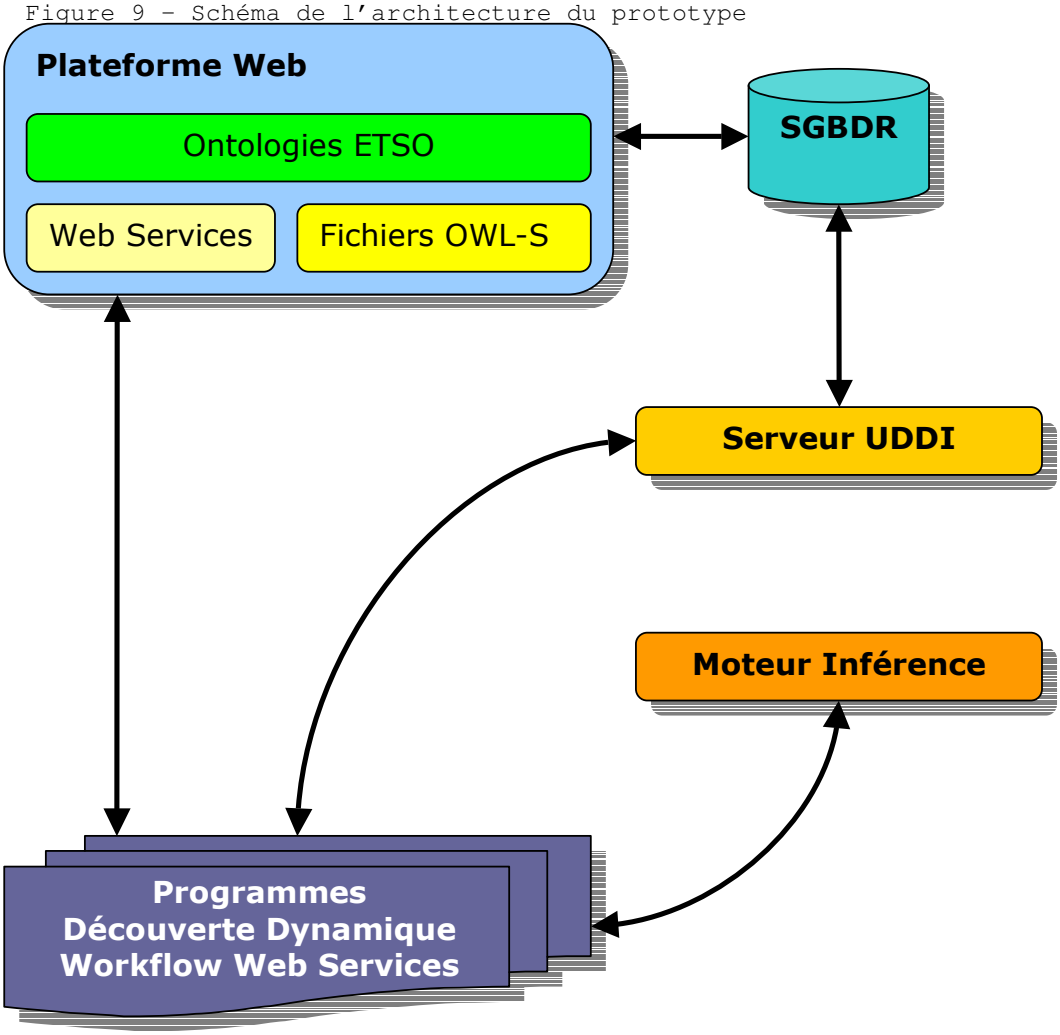
- **Système de Gestion de Base de Données**

  Un système de gestion de base de données relationnel était nécessaire pour le serveur UDDI même si une base de données XML aurait pu être envisagé. Cependant, le choix de stocker également les fichiers OWL-S dans une base de données relationnelle a renforcé l'idée d'utiliser un SGBDR. Deux bases de données se sont donc révélées nécessaires. La première pour gérer la base UDDI et exclusivement utilisé par le serveur UDDI. La deuxième pour stocker et délivrer les fichiers OWL-S.

- **Un programme client**

  Le programme, si possible avec une interface graphique, devait posséder la capacité de pré filtrer une liste de web services répertoriés sur un serveur UDDI. Puis, effectuer sur cette liste des web services, une découverte dynamique d'un workflow de web services.

Figure 9 – Schéma de l'architecture du prototype



Analyse des besoins techniques

L'ensemble des besoins techniques a été évalué sur les bases d'une logique open-source, ou, dans la configuration la moins favorable dans une logique de gratuité. Les solutions multi plateformes ont été aussi privilégiées afin de pouvoir facilement installer sur d'autres machines certains éléments clés de l'architecture logicielle, qui se prête fort bien d'ailleurs à une architecture distribuée.

- **Langage de développement**

  Le Web Sémantique et les outils qui lui sont associés sont majoritairement issus de la recherche universitaire, ce qui a signifié dans les faits que les apis disponibles où les outils proposés n'étaient utilisables qu'avec le langage java. L'universalité du langage java m'a permis d'intégrer et d'utiliser l'ensemble des différents éléments avec un seul et même langage. Cette universalité, outre l'aspect multi plateforme, s'est avéré un avantage substantiel lors de la phase de développement du prototype. La version du langage java utilisée lors du développement a été la version 1.4.2.

- **L'environnement de développement**

  Eclipse, de part le fort soutien de la communauté open source et à la base crée sur une initiative d'IBM, s'est imposé comme un environnement de développement multi-langages de première ordre. Sa capacité à être étendu à travers des plugins en fait un outil de choix lors d'un développement demandant l'utilisation et l'intégration de composants logiciels hétérogènes. Outre, sa capacité à être étendu, Eclipse propose une implémentation technique intéressante comme l'utilisation de SWT (Standard Widget Toolkit) afin de pallier au manque de performance des différentes solutions proposées par Sun.
  L'ensemble de l'affichage graphique du prototype a été fait en utilisant l'api SWT afin de garantir un affichage graphique avec des performances suffisante. Le prototype s'appuie aussi sur le framework graphique GED/Draw2D qui a été utilisé afin de réaliser le dessin du workflow sous forme de graphe orienté.
  Un plugin Eclipse Tomcat a été utilisé lors du projet, il s'agit du plugin de la société Sysdeo et la version était : 3.0 béta.
  La version d'Eclipse utilisée pendant l'ensemble du développement a été la version 2.1.3.

- **Editeur d'ontologies OWL**

  L'un des seuls éditeurs d'ontologies proposant le support d'OWL 1.0 est « protégé » de l'université de Stanford. Ecrit en java, il permet de construire son ontologie de manière relativement aisée en proposant une représentation de l'ontologie sous forme d'arbre. Un autre éditeur d'ontologies OWL a été utilisé pendant le développement du prototype, il s'agit de SWOOP de Mindswap (Maryland information and network dynamics lab semantic web agents project).

- **Plateforme d'hébergement des Web Services et des fichiers OWL et OWL-S**

  En matière de plateforme Web Services mon choix, s'est tout naturellement porté sur Axis proposé par la fondation Apache qui est une implémentation du protocole SOAP pour Java. Axis permet d'implémenter de manière transparente des web services écris en java au sein du serveur d'application J2EE Apache Tomcat et de les déployer. Il en résulte un plateforme web hybride supportant à la fois des requêtes HTTP et des requêtes SOAP.
  La plateforme se veut donc à la fois un serveur web délivrant des pages html dynamiquement au moyen de servlets et de jsp et une plateforme d'hébergement de web services.
  La version de Tomcat utilisée est la 5.0 et la version d'axis est la 1.1 1021 compilé avec la version de ant 1.6.1.

- **Serveur UDDI**

  Le serveur UDDI (Universal Description, Discovery and Integration) open source utilisé est « Juddi ». C'est une implémentation réalisé par la fondation apache. Juddi se présente sous la forme d'un ensemble de web services qui s'intègrent au serveur Tomcat/Axis et permettent d'administrer et d'utiliser un serveur Uddi. Juddi propose en standard de stocker l'ensemble des données sur un grand nombre de bases de données relationnelles ou bien sur une base xml.
  L'api java pour utiliser Juddi est UDDI4J qui est co-développé par HP et IBM.
  La version utilisée de Juddi est la 1.5.0-beta2-b51 et elle a été compilé avec la version de ant 1.6.1 et la version UDDI4J est 2.0.2.

- **Moteur d'inférence**

  Racer développé par Volker Haarslev, professeur associé à l'université de Concordia (Montreal Canada) et Ralf Möeller, professeur à l'université de science et technologie (Hambourg Allemagne) s'est vite imposé comme la solution la plus intéressante. Racer est le seul moteur d'inférence à disposer nativement de la possibilité de raisonner sur des ontologies au format OWL DL et d'être accessible de manière externe. Cela permet de limiter la complexité du client et grâce à cette approche modulaire de pouvoir facilement passer à une architecture distribuée. De plus, Racer se décline dans une version proxy qui autorise le load balancing et permet d'envisager la montée en charge. Racer, par sa maturité m'a paru le plus apte à être intégré dans le prototype sans compter que sa faculté à raisonner à la fois sur les TBox et les ABox est un atout de taille.
  Pellet, est le seul autre moteur d'inférence disponible spécifiquement pour OWL mais sa maturité ne m'a paru suffisante et il est encore en développement.
  Racer a été utilisé avec JRacer qui est un client TCP et qui est développé par l'Université de Sciences Appliqués de Wedel. Je n'ai pas retenu l'utilisation de HTTP car cela nécessitait d'utiliser DIG (Description Logic interface) et donc par conséquent de convertir les

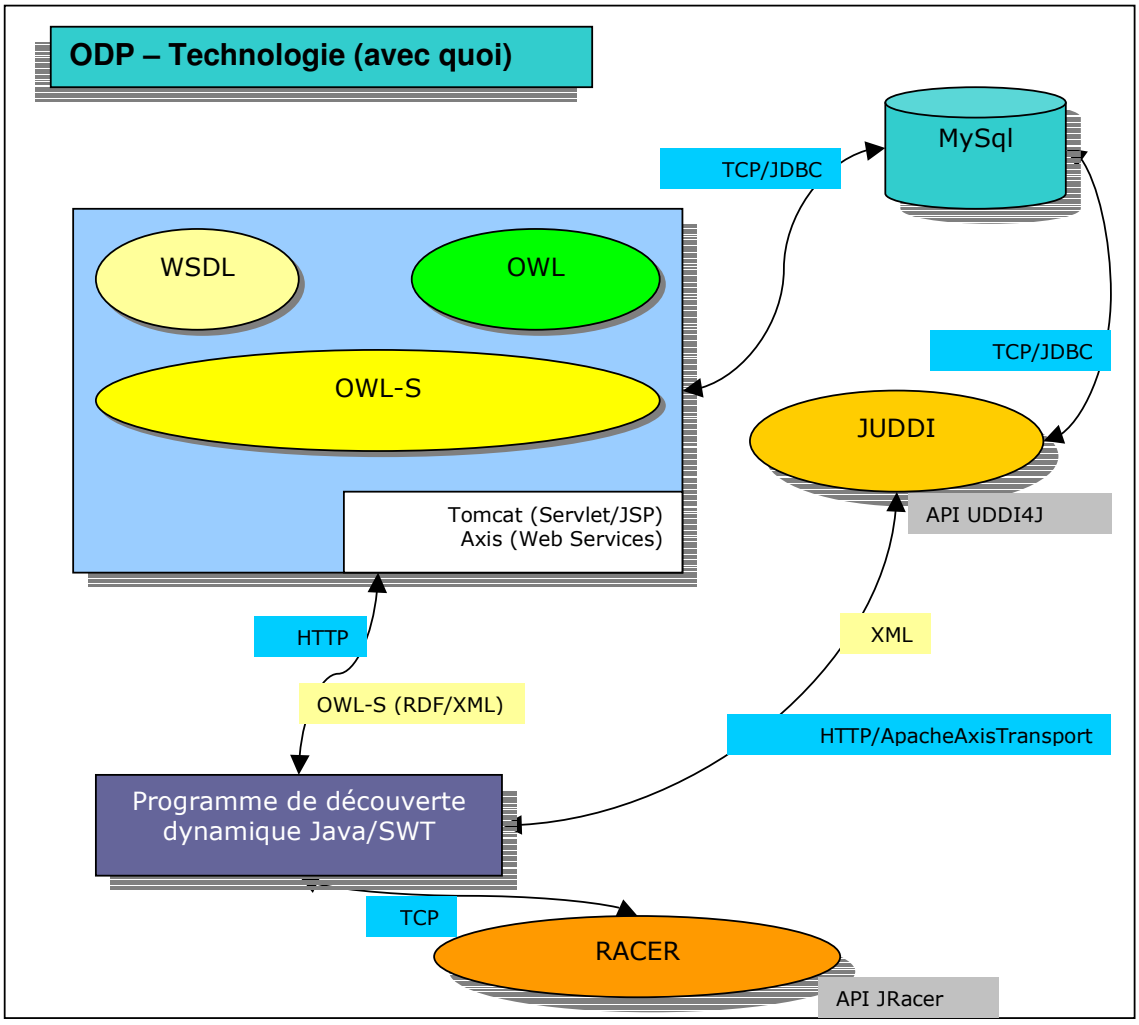
ontologies au format xml. Le principal inconvénient de Racer est qu'il n'est pas open source mais est gratuit dans le cadre d'une utilisation non commerciale. La version de Racer utilisée a évolué de la 1.7.15 à la version 1.7.21.

- **SGBD**

  Le stockage des fichiers OWL-S et de la base UDDI nécessitait un SGBDR. Mon choix s'est porté sur MySQL que je connais de longue date et qui a toujours répondu à mes besoins. Dans l'élaboration du prototype, la base de données n'était pas un élément critique aussi le principal pré-requis était de disposer d'une api JDBC. Le prototype utilise donc deux bases de données :
  - Edfwebsemantique qui est dédiée aux fichiers OWL-S.
  - Juddi qui est dédié au serveur UDDI.

  La version de MySQL était la 4.0 et l'api Jdbc utilisé a été MySQL Connector/J, version 3.0.14.

Figure 10 – Schéma ODP des technologies mises en oeuvre par le prototype



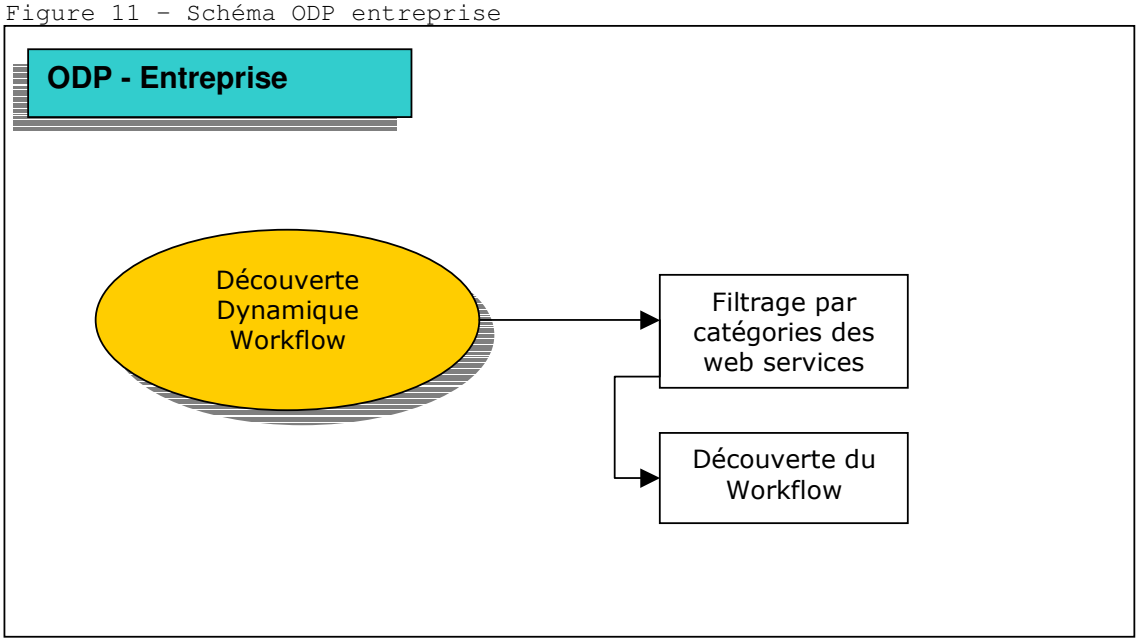
- **Fichiers OWL-S**

  Bien qu'une majorité du travail de création des fichiers OWL-S se soit faite manuellement, j'ai créé la structure de chacun des fichiers avec un petit programme de Mindswap nommé WSDL2OWLS. De même, la validation de la bonne conformité du fichier OWL-S, un fois dûment complété, s'est faite avec un autre programme de Mindswap appelé « OWL-S validation ».

ODP (Open Distributed Processing)

Sur la proposition de Laurent Olivry, ingénieur-chercheur nous avons procédé à l'analyse ODP du prototype. En effet, l'architecture du prototype se prêtait bien à une modélisation sous la forme ODP, c'est à dire à une architecture distribuée orienté objet. Le modèle ODP définit plusieurs points de vue pour spécifier des systèmes à des niveaux d'abstractions différents. Ces points de vue étant : l'entreprise, l'information, le traitement, l'ingénierie et la technologie.

Figure 11 – Schéma ODP entreprise



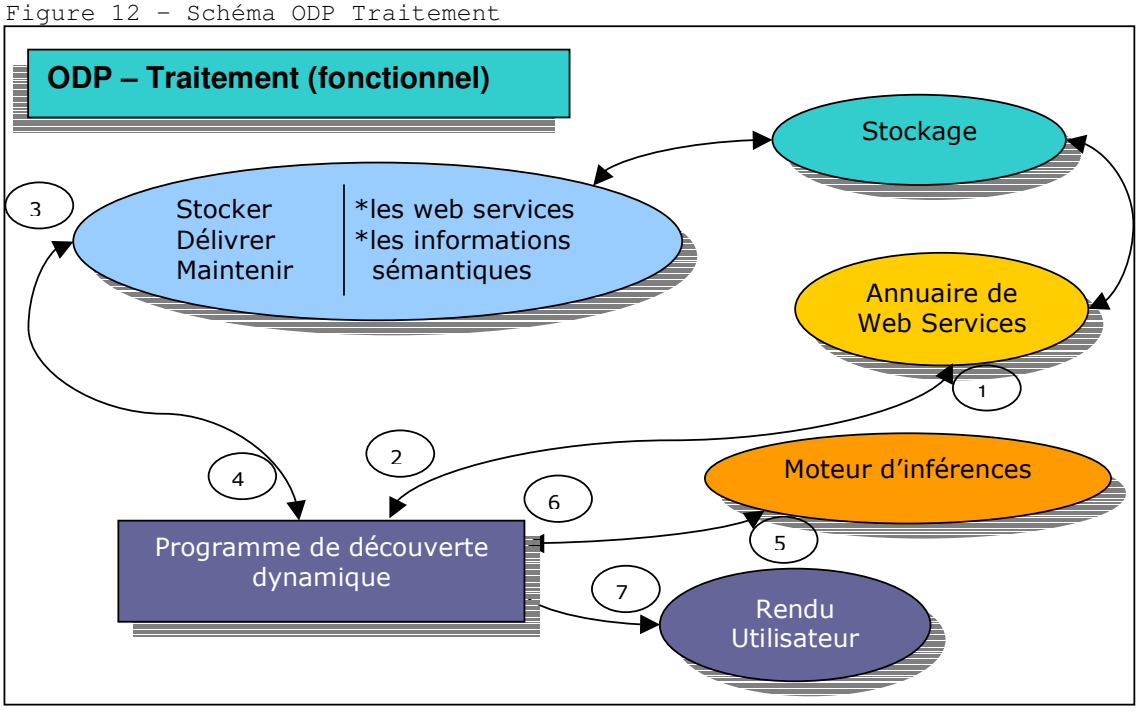
Le prototype permet à l'entreprise d'effectuer une découverte dynamique d'un workflow de web services. Deux tâches connexes sont nécessaires pour parvenir à ce résultat :

- Le filtrage par catégorie des web services qui permet de sélectionner un ensemble de web services appartenant à la même catégorie d'activité. Cette catégorie est un code NAICS qui est rattaché à chaque web service répertorié sur le serveur UDDI.

- La découverte du workflow qui tente de découvrir à partir de la liste des web services sélectionnés par leur code catégorie un enchaînement de

web services à partir des seules informations sémantiques attachées à chacun des web services.

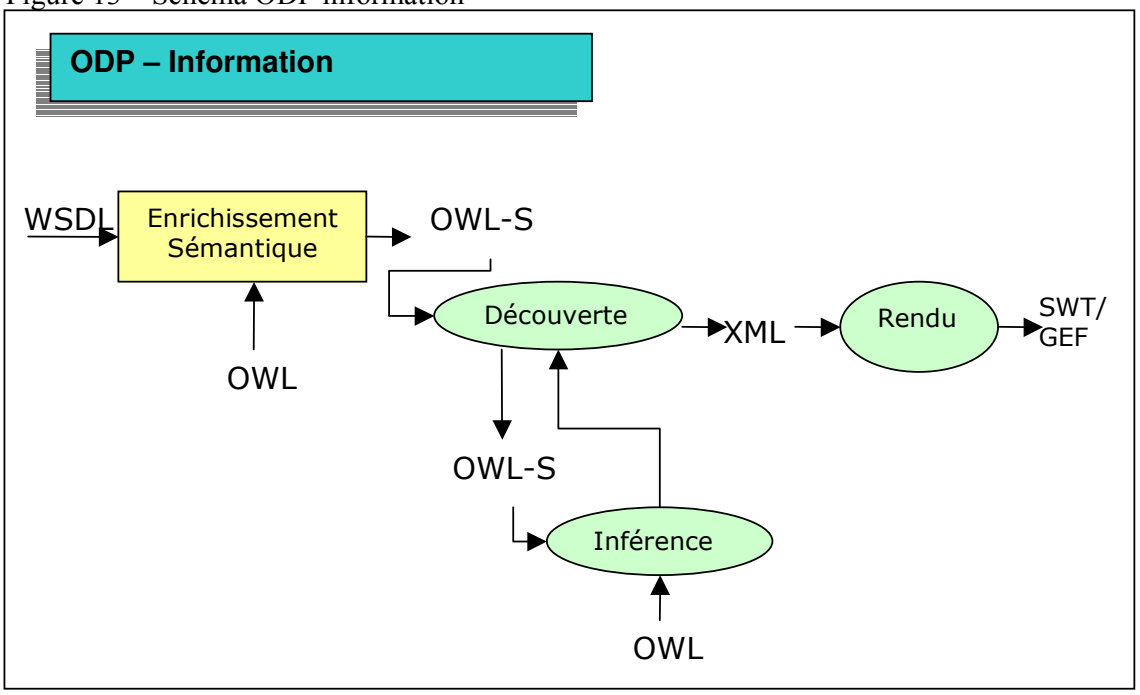
Figure 12 – Schéma ODP Traitement



Le traitement complet et l'interaction entre les différentes parties du prototype permet de dégager sept étapes :

- 1 – Interrogation de l'annuaire de web services (serveur UDDI) avec un code catégorie.
- 2 – Rapatriement et affichage de la liste de web services possédant un code catégorie correspondant. Les web services sont triés par sociétés.
- 3 – Les informations sémantique liées à chaque web service dans un fichier OWL-S sont demandées à partir de « l'access point » permettant de localiser un web service sur la plate forme web.
- 4 – Les informations sémantiques sont récupérées par le programme de découverte dynamique.
- 5 – Les informations sémantiques sont transmises au moteur d'inférences afin que celui-ci puisse déterminer les éventuelles connections sémantiques entre chaque web services.
- 6 – Le résultat de l'inférence entre deux web services est retourné programme de découverte dynamique.
- 7 – Après le parcours complet de l'ensemble des solutions possibles, un résultat graphique représentant la solution sous forme de graphe est affichée.

Figure 13 – Schéma ODP information

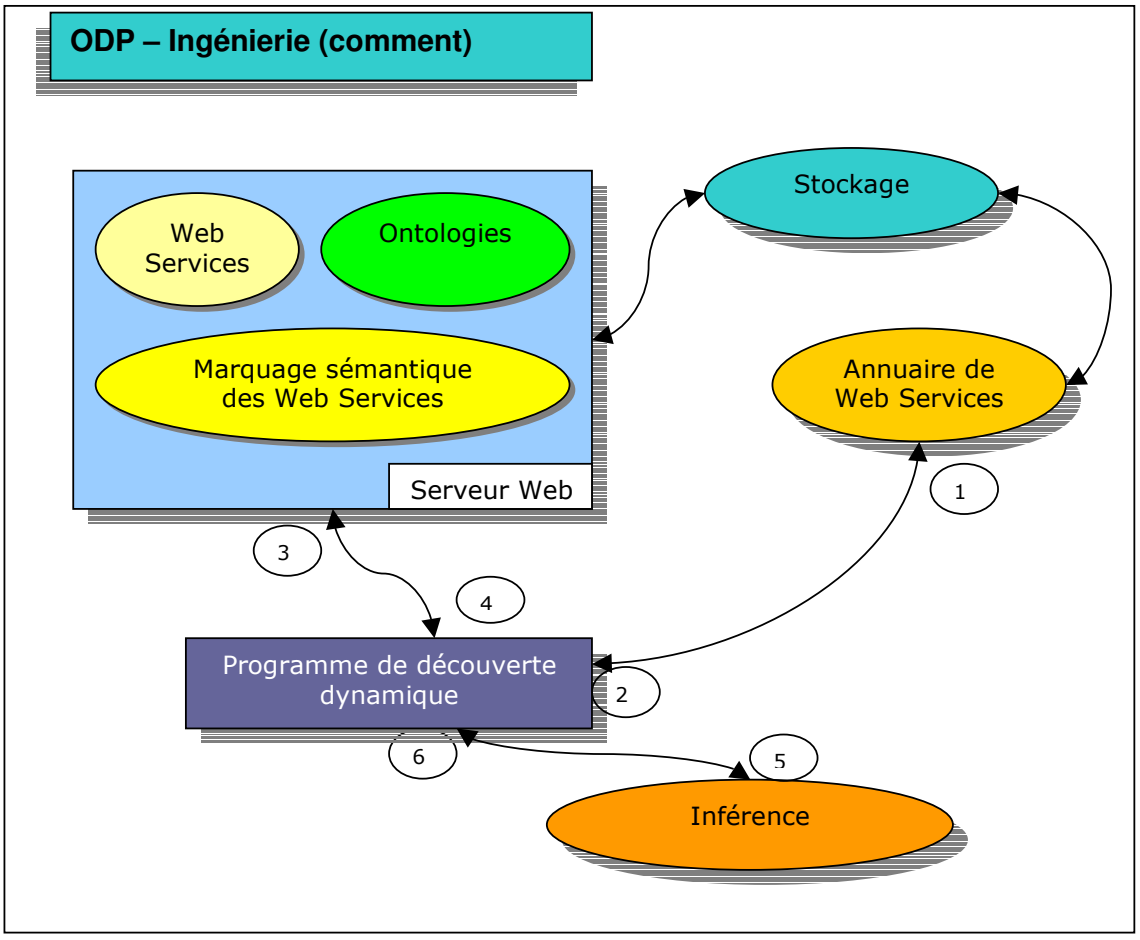


La transformation de l'information au cours des différentes étapes de traitement au sein du prototype permet de se rendre compte que l'information est modifiée et transformée au cours de deux grandes étapes :

- La première étape que l'on pourrait qualifier d'étape de type pré processing permet à partir de l'information purement technique du WSDL décrivant un web service d'effectuer un « enrichissement sémantique » en rattachant des concepts provenant des deux ontologies au web service. Cela permet de catégoriser sémantiquement le web service et ainsi de synthétiser la connaissance technique et sémantique au sein d'un même document OWL-S.

- La deuxième étape montre que l'information sémantique issue de la première étape est utilisée lors de la phase d'inférence appliquée à chacun des web services. Cette phase nécessite l'apport d'informations sémantique «extérieure». En effet, l'inférence ne peut se faire que lorsque le moteur a récupéré les différentes ontologies nécessitées par les clauses « import » des ontologies OWL et fichiers OWL-S.
  A chaque nouveau web service découvert un fichier XML représentant l'arborescence hiérarchique du graphe solution est complété. Les informations contenues dans le fichier XML seront utilisées par le moteur de rendu du prototype afin de représenter le workflow découvert sous forme de graphe ou chaque nœud représente un web service.
  Une des évolutions possible du prototype serait sans doute d'utiliser le fichier XML représentant la solution workflow pour générer un fichier BPEL en utilisant par exemple une transformation XSL.

Figure 14 – Schéma ODP ingénierie

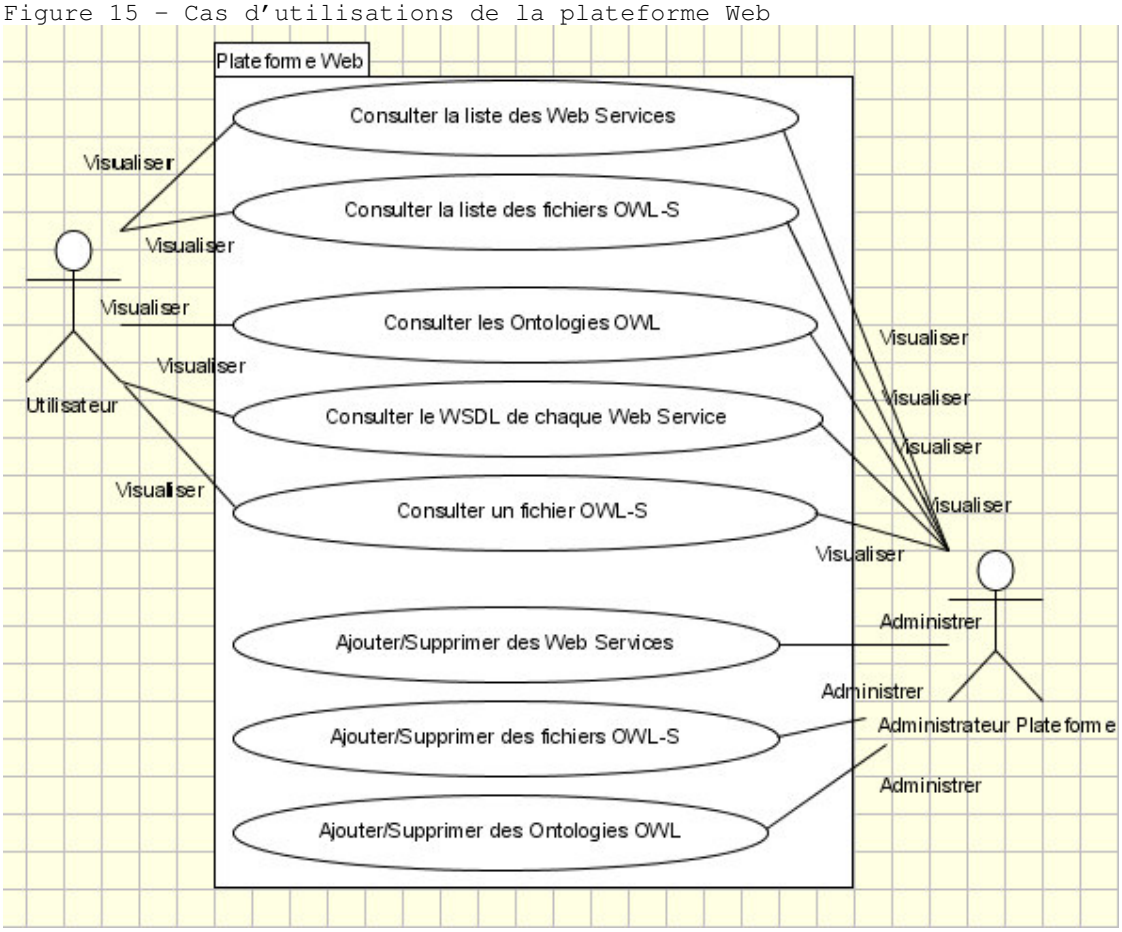


Cas d'utilisations

Les cas d'utilisations illustrent les fonctionnalités principales de la plateforme Web et du programme de découverte dynamique d'un workflow de web services.

Plateforme Web

Figure 15 – Cas d'utilisations de la plateforme Web



Acteur Utilisateur

- Consulter la liste des Web Services
  L'utilisateur peut consulter la liste des Web Services répertoriés et invocables sur la plateforme.

- Consulter le WSDL de chaque Web Service
  Chaque Web Service est défini par un WSDL qui permet un consommateur de ce web service d'être renseigné sur la façon d'invoquer celui-ci. Pour accéder au WSDL, il faut rajouter « ?wsdl » à la fin de l'url localisant le web service.

- Consulter la liste des fichiers OWL-S
  L'utilisateur peut consulter la liste des fichiers OWL-S disponibles.

- Consulter un fichier OWL-S
  Chaque fichier OWL-S peut être visualisé. Pour accéder à l'OWL-S, il faut rajouter « ?owls » à la fin de l'url localisant le web service.

- Consulter les ontologies OWL
  La plateforme héberge un ensemble d'ontologies au format OWL et autorise leur consultation.

Acteur Administrateur de la plateforme Web

- L'acteur Administrateur peut effectuer l'ensemble des actions de l'acteur Utilisateur. Il peut aussi effectuer un certain nombres de tâches spécifiques à la maintenance de la plateforme web.

- Ajouter/Supprimer des web services
  L'administrateur peut ajouter, modifier ou supprimer un web service sur la plateforme.

- Ajouter/Supprimer des fichiers OWL-S
  L'administrateur peut ajouter, modifier ou supprimer des fichiers OWL-S.

- Ajouter/Supprimer des ontologies OWL
  L'administrateur peut ajouter, modifier ou supprimer des ontologies OWL.

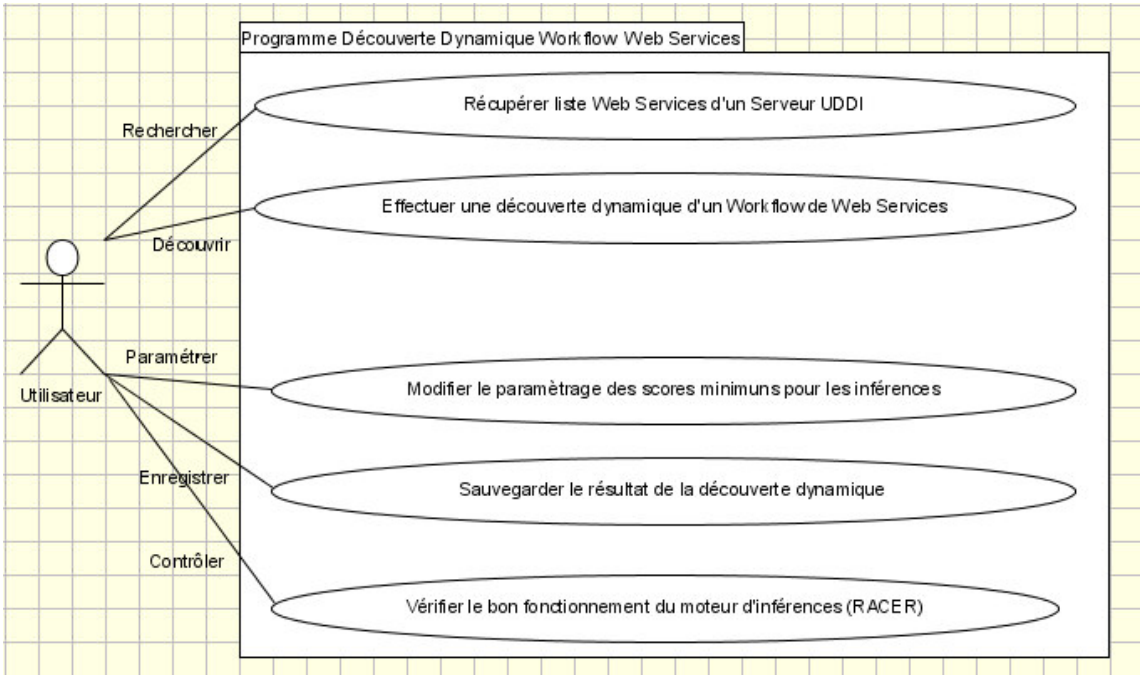
Programme de découverte dynamique d'un workflow de Web Services

Acteur Utilisateur

- Récupérer la liste des Web Services d'un serveur UDDI
  L'utilisateur peut effectuer une recherche d'une liste de web services référencés sur un serveur UDDI à partir d'un code NAICS (North American Industry Classification System).

- Effectuer une découverte dynamique d'un workflow de web services
  A partir d'une liste de web services, l'utilisateur peut demander à effectuer une découverte dynamique d'ordonnancement de web services. Pour cela, il faudra que l'utilisateur désigne le web service de départ.

Figure 16 – Cas d'utilisations du programme de découverte dynamique d'un workflow de Web Services



- Modifier le paramétrage des scores minimums pour les inférences.
  L'utilisateur peut modifier à sa guise le score minimum d'intermédiation pour les concepts rattachés aux web services ainsi que pour les concepts liés aux paramètres.

- Sauvegarder le résultat de la découverte dynamique
  Le résultat de la découverte de l'ordonnancement des web services est stocké dans un fichier XML qui peut être sauvegardé sur disque.

- Vérifier le bon fonctionnement du moteur d'inférences (RACER)
  L'utilisateur peut vérifier que RACER est présent sur le réseau et prêt à fonctionner.

Développement logiciel

Avant d'effectuer le développement de la plateforme web et du programme client de découverte de workflow, il a d'abord fallu développer un programme (UddiPublisher) permettant de déployer les web services sur le serveur UDDI.

Déploiement des Web Services ETSO sur le Serveur UDDI

WSDL ou Web Service Description Language est un schéma XML qui permet de décrire les web services. Il permet de séparer la description des fonctionnalités abstraites offertes par un service, des détails concrets d'une description de service, tels que "comment" et "où" cette fonctionnalité est proposée. C'est donc un langage décrivant les fonctionnalités abstraites d'un service ainsi que l'architecture décrivant les détails concrets de la description de service. Le WSDL décrit quatre ensembles de données importants :

- Information d'interface décrivant toutes les fonctions disponibles publiquement.
- Information de type de donnée pour toutes les requêtes de message et requêtes de réponse.
- Information de liaison sur le protocole de transport utilisé.
- Information d'adresse pour localiser le service spécifié.

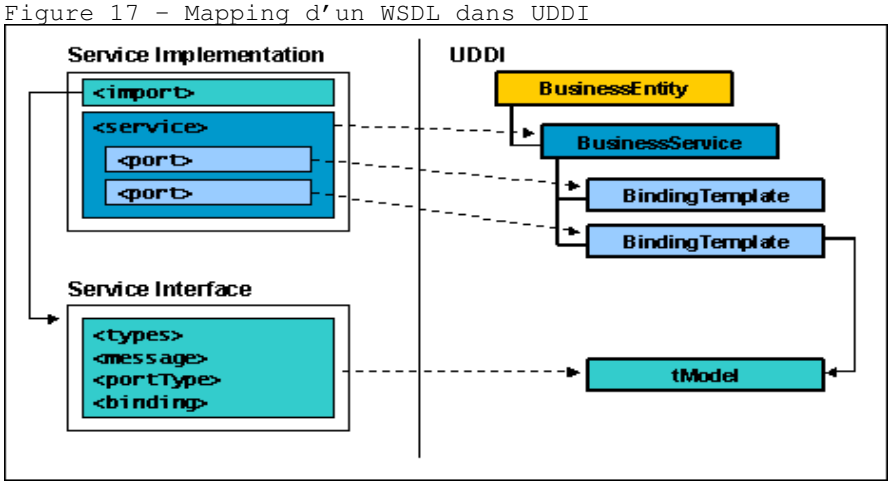
UDDI fournit une méthode pour publier et rechercher ces descriptions de web services. UDDI fournit un support pour trouver à la fois des informations sur l'entreprise qui fournit des services et les dit services.
Il y a quatre types de données primaire dans un registre UDDI :

- BusinessEntity qui fournit des informations sur l'entreprise proposant un ou plusieurs services.
- BusinessService qui contient des informations sur les services proposés.
- BindingTemplate qui contient les informations techniques à propos de l'access point de chacun des web services.
- tModel qui contient les descriptions des spécifications.

Pour publier et rechercher des descriptions de service WSDL dans un serveur UDDI, les documents WSDL sont divisés en deux parties :

- « Service implementation » qui décrit les instances d'un service. Chaque instance est définie en retenant l'élément « service » du WSDL qui sera utilisé pour publier le BusinessService.
- « Service interface » est publié comme un tModel qui sera lui-même publié par le provider du « service interface » à savoir le BindingTemplate.

Figure 17 – Mapping d'un WSDL dans UDDI



L'ensemble des web services peuvent appartenir à une ou plusieurs catégories (CategoryBag). Cette catégorie permettra d'effectuer une recherche sur une catégorie précise de web services et ainsi effectuer un premier préfiltrage. Pour l'ensemble des web services appartenant à l'ESS, j'ai utilisé un code appartenant à la nomenclature NAICS (North American Industry Classification System). Le code catégorie utilisé était le 22112 (Electric power transmission, control, and distribution).

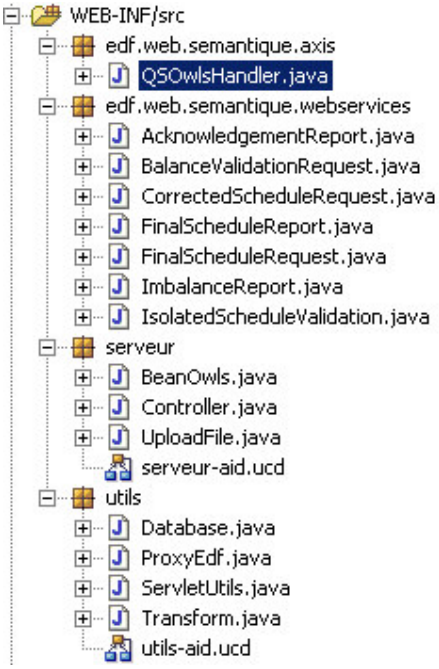
Extrait de la classe Uddi.java du programme UddiPublisher. Méthode «publishServiceImplementation» qui permet de publier un web service.

```
public BusinessService publishServiceImplementation(
        String businessServiceName,
        String businessKey,
        String tModelKey,
        String accessPointURI,
        String wsdlImplURL,
        String categoryBagName,
        String categoryBagValue)
        throws Exception {

        ServiceDetail serviceDetail;
        Vector tModelList = new Vector();

        TModelInstanceInfo tModelInstanceInfo;
        TModelInstanceDetails tModelInstanceDetails = new TModelInstanceDetails();

        Vector tModelInstanceInfoList = new Vector();
        InstanceDetails instanceDetails = new InstanceDetails();
        OverviewDoc overviewDoc = new OverviewDoc();

        //BusinessService
        BusinessService businessService = new BusinessService();
        businessService.setBusinessKey(businessKey);
        businessService.setDefaultNameString(businessServiceName, "en");
        businessService.setDefaultDescriptionString("");

        //bindingTemplate
        BindingTemplate bindingTemplate = new BindingTemplate();
        bindingTemplate.setDefaultDescriptionString("");
        AccessPoint accessPoint = new AccessPoint(accessPointURI, "http");
        bindingTemplate.setAccessPoint(accessPoint);
        //Création du tModelInstanceInfo en utilisant la tModelKey
        tModelInstanceInfo = new TModelInstanceInfo(tModelKey);

        //overview URL
        OverviewURL overviewURL = new OverviewURL(wsdlImplURL);
        overviewDoc.setOverviewURL(overviewURL);
        instanceDetails.setOverviewDoc(overviewDoc);
        tModelInstanceInfo.setInstanceDetails(instanceDetails);
        tModelInstanceInfoList.addElement(tModelInstanceInfo);
        tModelInstanceDetails.setTModelInstanceInfoVector(tModelInstanceInfoList);

        //ajoute le tModelInstanceDetails au binding template
        bindingTemplate.setTModelInstanceDetails(tModelInstanceDetails);
        BindingTemplates bindingTemplates = new BindingTemplates();
        Vector bindingTemplateVector = new Vector();
        bindingTemplateVector.addElement(bindingTemplate);
        bindingTemplates.setBindingTemplateVector(bindingTemplateVector);
        businessService.setBindingTemplates(bindingTemplates);
        CategoryBag categoryBag = new CategoryBag();
        Vector krList = new Vector();
        KeyedReference kr = new KeyedReference(categoryBagName, categoryBagValue);
        kr.setTModelKey(taxonomyModelKey);
        krList.add(kr);
        categoryBag.setKeyedReferenceVector(krList);
        businessService.setCategoryBag(categoryBag);
        Vector businessServiceVector = new Vector();
        businessServiceVector.addElement(businessService);

        //Sauve le tModel
        serviceDetail = proxy.save_service(authToken.getAuthInfoString(),
        businessServiceVector);
        return (BusinessService) serviceDetail.getBusinessServiceVector().elementAt(0);
}
```

Plateforme Web

Le développement web s'est concentré sur le développement de servlets et de jsp en utilisant le pattern mvc2 et à faciliter l'intégration des web services au sein de la plateforme. Le développement du plugin pour axis s'est aussi révélé important.

La plateforme s'organise en 4 packages différents :



- edf.web.semantique.axis
  Ce package contient la classe qui étend les mots clés d'Axis et permet de disposer de l'extension «?owls ».

- edf.web.semantique.webservices
  Ce package regroupe l'ensemble des web services participant à une transaction ESS ETSO. Les web services ont été développé sans utiliser JWS (Java Web Service) afin de pouvoir disposer d'une plateforme où servlets, jsp et web services sont très intégrés. L'ensemble du déploiement des web services s'est donc faites manuellement. (Voir en annexe, les fichiers de configuration server-config.wsdd et web.xml).

- serveur
  Ce package contient la servlet «controller » qui gère la cinématique de la navigation sur le site en utilisant un ensemble de jsp. Il contient aussi un bean dédié aux fichiers OWL-S. UploadFile est une servlet permettant de télécharger des fichiers dans la base de données sous forme de Blob.

- utils
  Ce package regroupe différentes classes comme celle gérant l'accès à la base de donnée avec jdbc, la gestion du proxy edf ainsi que des classes facilitant les sérialisations de toutes natures.

Extrait du bean BeanOwls. Méthode getOwlsFileFromDb qui récupère un fichier OWL-S depuis la base de données.

```
/**
 * Récupère un fichier OWL-S depuis la database
 *
 * @param ontologyName
 * @return InputStream
 */
public InputStream getOwlsFileFromDb(String ontologyName) {
        String sQuery = null;
        InputStream result = null;
```

```
        Database db = new Database();
        connection = db.getConnection(0);

        try {
                sQuery = "select ontology_Data from ontology where ontology_Name = '" +
                ontologyName + "'";
                PreparedStatement ps = connection.prepareStatement(sQuery);

                rs = ps.executeQuery();

                if (rs.next()) {
                        Blob agent = rs.getBlob("ontology_Data");
                        result = agent.getBinaryStream();
                        System.out.println("inputstream=" + result.toString());
                }
        } catch (SQLException e) {
                e.printStackTrace();
        }

        db.disconnect();

        return result;
}
```

Récupération des fichiers OWL-S

Le mécanisme de récupération des informations sémantiques associées au web service, autrement dit, du fichier OWL-S fonctionne de la même façon que celui permettant de récupérer le fichier WSDL lié à un web service.

En effet, de même qu'il suffit d'ajouter à la fin de l'URL pointant sur web service, la partie suivante «?wsdl» pour que l'on récupère le fichier WSDL associé au web service.

Il faut préciser que si la manière est identique, le WSDL peut être dynamiquement généré tandis que le fichier OWL-S lui fait l'objet d'une implémentation manuelle préalable.

Axis qui est déployé dans le serveur d'application Tomcat gère l'ensemble des accès en rapport aux web services et en permet l'exécution. Il s'occupe notamment de décoder des messages SOAP entrant, d'appeler le web service désigné et d'encoder le résultat retourné par le web service toujours au format SOAP.

Pour permettre l'invocation automatique d'un fichier OWL-S, il faut donc créer, un plugin pour Axis ou plus précisément un plugin gérant une nouvelle extension de chaîne de caractères. Axis, fournit nativement trois chaînes : ?wsdl, ?version et ?method.

Le nouveau plugin permettra donc de gérer : ?owls. Chaque plugin implémente l'interface : *org.apache.axis.transport.http.QSHandler* qui définie une méthode « invoke ». A chaque fois, que l'extension ?owls sera demandée, la méthode « invoke » sera exécutée. Un paramètre « *MessageContext* » fournit notamment le nom du web service demandé.

La classe QSOwlsHandler (package edf.web.semantique.axis) a donc été créée et gère l'extension ?owls. Une fois le nom du web service obtenu, le fichier OWL-S correspondant est récupéré à partir de la base de données sous forme de flux représentant un Blob (Binary Large Object) et est copié dans le flux réponse.

Extrait de la méthode invoke de la classe QSOwlsHandler :

```
//récupère le nom du web service
owlsName = (String) msgContext.getProperty(HTTPConstants.PLUGIN_SERVICE_NAME);


//Récupère le fichier owl-s de la database
if (owlsName != null) {
    BeanOwls ontology = new BeanOwls();
    InputStream in = null;
    in = ontology.getOwlsFileFromDb(owlsName);
    if (in != null) {
           try {
                  while ((c = in.read()) != -1) {
                         out.write(c);
                         }
                  } catch (IOException e) {
                         e.printStackTrace();
                         response.setStatus(HttpURLConnection.HTTP_INTERNAL_ERROR);
                         response.setContentType("text/html");
                  }
           }
    }
}
```

Pour que le plugin soit déployé, il faut le déclarer dans la section transport du descripteur de déploiement «server-config.wsdd ».

Extrait du fichier server-config.wsdd :

```
<transport name="http">
  <parameter name="qs.owls" value="edf.web.semantique.axis.QSOwlsHandler"/>
  <requestFlow>
   <handler type="URLMapper"/>
   <handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
  </requestFlow>
 </transport>
```

Figure 18 – Capture d'écran de la plateforme Web (liste des fichiers OWL-S)

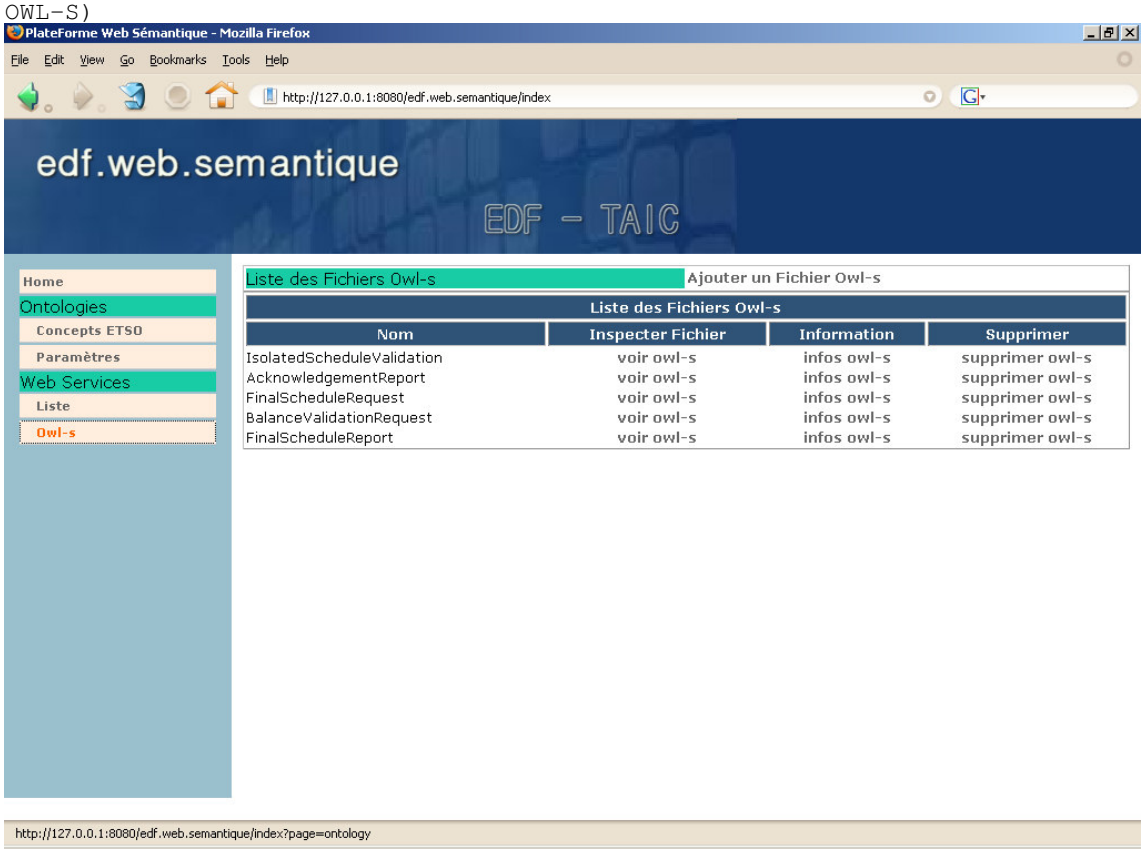


Figure 19 – Diagramme de Classes du package utils

Figure 20 – Diagramme de Classes du package serveur

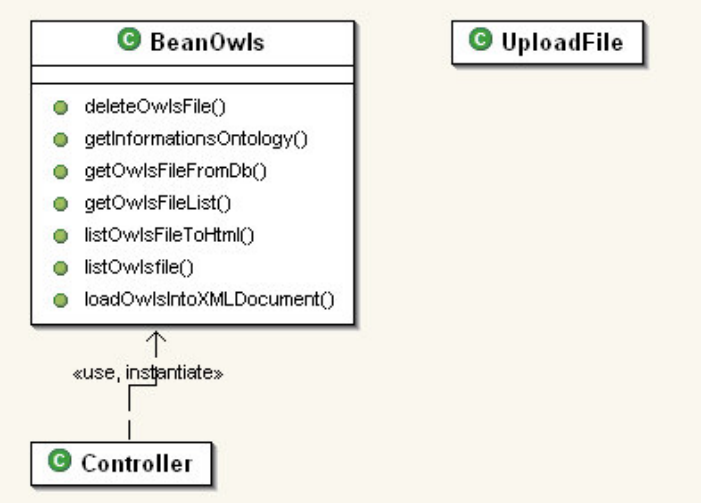
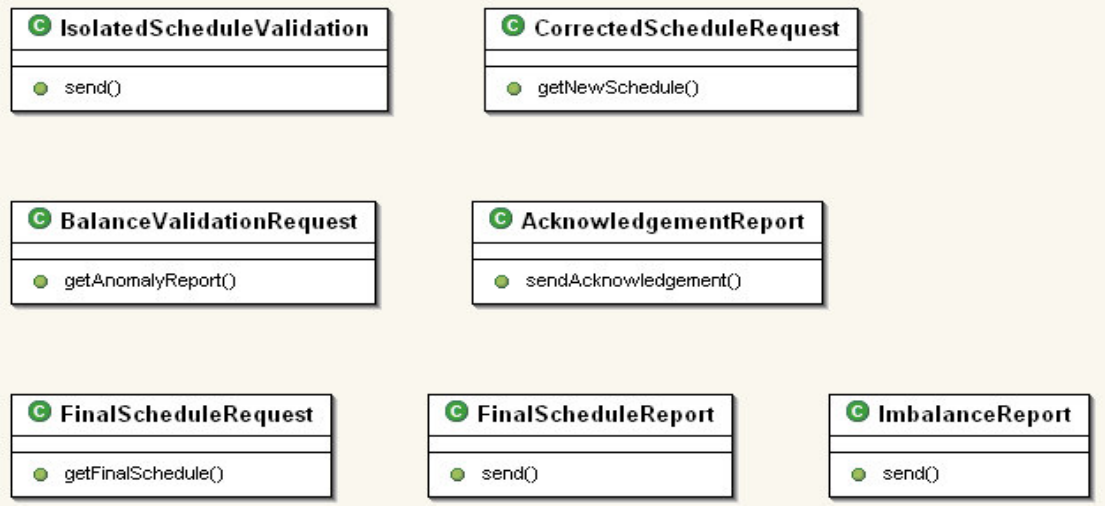


Figure 21 – Diagramme de Classes du package edf.web.semantique.webservices



Programme de découverte dynamique d'un workflow de Web Services

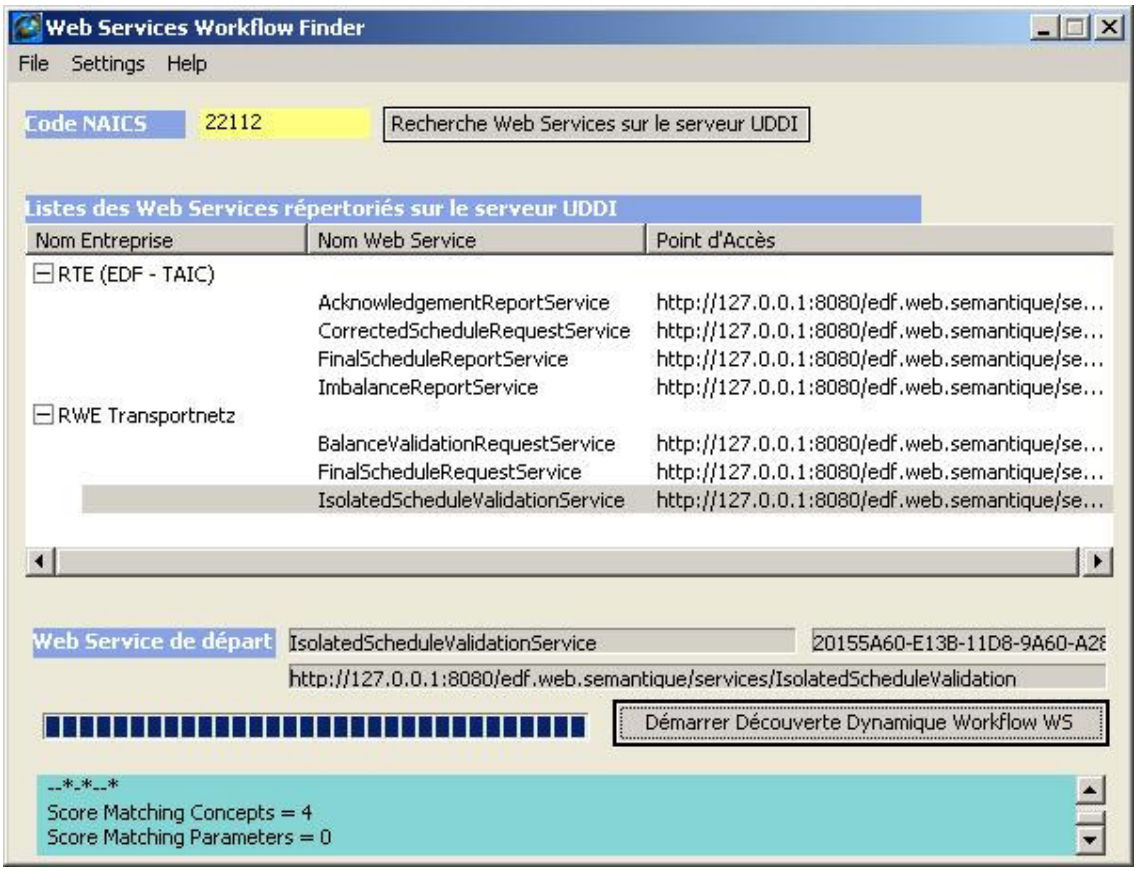
Le programme de découverte dynamique se présente sous la forme d'un programme (jar exécutable) pouvant être exécuté de manière autonome. Le programme est doté d'une interface graphique (SWT) afin de rendre son utilisation plus conviviale et plus ergonomique.

Le programme permet d'effectuer des sélections à partir de code catégories, et ainsi obtenir, un ensemble de web services référencés sur un serveur UDDI. Seuls les web services sélectionnés pourront participer au processus de découverte dynamique du workflow.

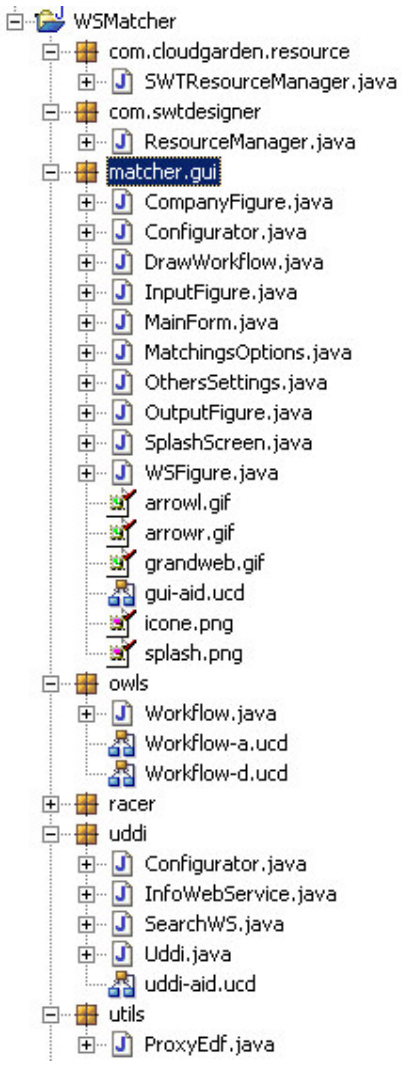
Le développement s'est articulé autour de trois phases principales :

- La présélection des web services référencés sur le serveur UDDI.
- La découverte dynamique du workflow.
- La représentation graphique du workflow.

Figure 22 – Capture d'écran du programme en cours de processus de découverte.



Le programme est composé de 7 packages :



- com.cloudgarden.resource
  Package contenant la classe SWTResourceManager qui gère les ressources SWT.

- com.swtdesigner
  Package contenant la classe ResourceManager qui gère les ressources du système d'exploitation associées aux controles SWT.

- matcher.gui
  Package contenant toutes les classes utilisant des ressources graphiques SWT.

- owls
  Package contenant la classe Workflow. Cette classe contient l'ensemble des méthodes permettant de réaliser la découverte dynamique du workflow.

- racer
  Package qui contenant l'ensemble des classes permettant de s'interfacer avec le moteur d'inférences RACER.

- uddi
  Package contenant l'ensemble des classes permettant de s'interfacer et d'interroger le serveur UDDI.

- utils
  Package contenant la classe ProxyEdf qui permet d'utiliser le proxy d'EDF.

Présélection des web services référencés sur le serveur UDDI

La sélection d'un ensemble de web services correspondant à une catégorie bien précise se base sur une interrogation d'un serveur UDDI en utilisant un code catégorie ou category bag. L'api UDDI4J permet de récupérer un ensemble de web services à travers la fonction « findservice ». Celle ci permet de définir les critères qui seront pris en compte pour effectuer la recherche. Ici, seul le category bag sera utilisé. De l'ensemble des renseignements associés à chaque web services le plus important est son access point. L'access point définit l'url correspondant à la localisation du web service sur la plateforme.
Cet access point sera utilisé notamment pour récupérer le fichier OWL-S qui est associé à chaque web service.
Une fois, la liste des web services retournée par le serveur UDDI, le programme la triera par sociétés puis par ordre alphabétique et sera finalement affiché dans un composant SWT TableTree.

Extrait de la classe UDDI. Méthode getBusinessServiceByCategoryBag qui interroge la base UDDI et récupère le résultat.

```
/**
 * Recherche un ensemble de web services par code catégorie
 *
 * @param categoryBagName
 * @param categoryBagValue
 * @param maxRows
 *
 * @return Vector
 */
public Vector getBusinessServiceByCategoryBag(
                        String categoryBagName,
                        String categoryBagValue,
                        int maxRows) {
        Vector result = null;
        try {
                ServiceList serviceList = null;

                CategoryBag categoryBag = new CategoryBag();
                Vector krList = new Vector();
                KeyedReference kr = new KeyedReference(categoryBagName,
                categoryBagValue);
                kr.setTModelKey(taxonomyModelKey);
                krList.add(kr);
                categoryBag.setKeyedReferenceVector(krList);

                //récupère l'ensemble des services correspondant à ce category Bag
                serviceList = proxy.find_service(null, null, categoryBag, null, null,
                maxRows);

                Vector serviceInfoVector =
                serviceList.getServiceInfos().getServiceInfoVector();

                if (serviceInfoVector.size() != 0 )
                        result = new Vector();
                else
                        return result;

                //parcours l'ensemble des services et remplit un vecteur
                for (int i = 0; i < serviceInfoVector.size(); i++) {
                        ServiceInfo serviceInfo = (ServiceInfo)
                        serviceInfoVector.elementAt(i);

                        ServiceDetail serviceDetail =
                        proxy.get_serviceDetail(serviceInfo.getServiceKey());
                        Vector v = serviceDetail.getBusinessServiceVector();

                        BusinessService bs = (BusinessService) v.elementAt(0);
                        BindingTemplates bts = bs.getBindingTemplates();
```

```
                BindingTemplate bt = bts.get(0);
                AccessPoint ap = bt.getAccessPoint();

                String[] arrS = new String[4];
                arrS[0] =
                getBusinessEntity(serviceInfo.getBusinessKey()).getDefaultNameSt
                ring();
                arrS[1] = serviceInfo.getDefaultNameString();
                arrS[2] = serviceInfo.getServiceKey();
                arrS[3] = bt.getAccessPoint().getText();

                result.addElement(arrS);
            }
        } catch (UDDIException e) {
            e.printStackTrace();
        } catch (TransportException e) {
            e.printStackTrace();
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        return result;
}
```

Découverte dynamique du workflow

La découverte dynamique du workflow passe tout d'abord par la sélection d'un web service de « départ » qui permettra d'amorcer le processus en disposant des concepts associés à ce web service et au paramètre de sortie.
Pour cela, il faut tout d'abord récupérer du fichier OWL-S qui est associé à chaque web service en utilisant comme adresse de localisation de ce fichier, l'access point augmenté de l'extension « ?owl-s». L'api OWL-S permettant ensuite de charger ce fichier et d'extraire les différentes données sémantiques nécessaires à la réalisation de l'intermédiation entre deux web services.

(Rappel : le diagramme d'activité de l'intermédiation entre web services est situé page 25, figure 7)

Extrait de la classe Workflow. Extrait de la méthode «processMatching » qui permet de récupérer le fichier OWL-S.

```
OWLSReader lreader = OWLSFactory.createOWLSReader();
Service lservice = null;
try {
        lservice = reader.read(URI.create(infoWebService.getAccessPoint() +
        "?owls"));
} catch (Exception e) {
        textReport.append("\n Impossible de trouver le fichier Owl-S pour " +
        infoWebService.getWsName());
        System.out.println("Impossible de trouver le fichier owls pour " +
        infoWebService.getWsName());
        if (debugMode) e.printStackTrace();
                lservice = null;
}
```

Une fois le fichier OWL-S récupéré, le concept associé au web service est extrait de la partie profile du fichier ainsi que l'URI localisant l'ontologie. Une fois, ces deux éléments récupérés, l'inférence entre les concepts peut-être effectuée en utilisant le moteur d'inférences. Racer procédera alors à l'évaluation de la distance entre les concepts à partir de l'utilisation de relations de subsumptions.

La première chose à effectuer est le formatage des concepts afin que RACER puisse les utiliser. Les concepts doivent être formatés de la façon suivante : |#concept| .
L'intermédiation en elle-même consiste à interroger Racer, en lui passant les deux concepts et l'URI pointant sur l'ontologie et référençant à priori au moins un des deux concepts.
Racer fournit nativement, un certain nombre d'instructions permettant de tester :

- Une relation d'équivalence.
- Une relation de subsumption.

Il est à préciser que la subsumption inverse se détermine en utilisant la même instruction que pour la subsumption (ConceptSubsumesQ) mais en inversant les deux concepts.

Extrait de la classe Racer. Méthode «matchConcepts» qui détermine le degré de subsumption entre deux concepts.

```
/**
 * Effectue un matching sur des concepts
 *
 * @param concept1
 * @param concept2
 * @param uriOntology uri de l'ontologie sur laquelle on effectue l'inférence
 *
 * @return int correspondant au degrè de satisfaction du matching
 */
public static int matchConcepts(String concept1, String concept2, String uriOntology) {
	RacerClient racerClient = null;
	boolean racerResult = false;
	StringBuffer c1 = new StringBuffer();
	StringBuffer c2 = new StringBuffer();

	if ( (concept1 == null) || (concept2 == null) )
		return MatchingConstants.FAIL;

	racerClient = getConnectionToRacer(uriOntology);

	if (racerClient == null) return MatchingConstants.ABORT;

	//formatage pour racer
	c1.append("|#"); c1.append(concept1); c1.append("|");
	c2.append("|#"); c2.append(concept2); c2.append("|");

	if (debugMode) System.out.println("c1=" + c1 + " -- c2="+ c2);

	//concepts équivalents ?
	try {
		racerResult = racerClient.conceptEquivalentQ(c1.toString(),
		c2.toString());
		if (racerResult) return MatchingConstants.MATCH;
	} catch (IOException e1) {
		e1.printStackTrace();
	} catch (RacerException e1) {
		e1.printStackTrace();
	}

	//c1 subsumes c2 ?
	try {
		racerResult = racerClient.conceptSubsumesQ(c1.toString(),
		c2.toString());
		if (racerResult) return MatchingConstants.SUBSUMES;
	} catch (IOException e) {
		e.printStackTrace();
	} catch (RacerException e) {
		e.printStackTrace();
	}

	//subsumes inverse : c2 subsumes c1 ?
	try {
		racerResult = racerClient.conceptSubsumesQ(c2.toString(),
		c1.toString());
		if (racerResult) return MatchingConstants.SUBSUMES_INVERT;
	} catch (IOException e) {
		e.printStackTrace();
	} catch (RacerException e) {
```

```
            e.printStackTrace();
        }
        return MatchingConstants.FAIL;
}
```

Si le concepts sont jugés suffisamment proche en regard du paramétrage effectué par l'utilisateur, le programme compare alors les concepts liés à chacun des paramètres en entrée avec le concept en sortie du web service précédent.
Au final, si un des paramètres en entrée possède une distance sémantique probante à travers le concept qui lui est associé, toujours en regard au paramétrage de l'utilisateur, alors le web service est sélectionné comme appartenant au workflow. Ce web service s'insérera séquentiellement juste après afin de garantir une exécution du workflow.

Le nouveau web service découvert sera à son tour utilisé comme web service de départ et soumis à une intermédiation avec l'ensemble des web services ramenés par la présélection faite sur le serveur UDDI.

Cependant, le nouveau web service découvert sera testé pour déterminer si le concept associé à son paramètre de sortie correspond à un des paramètres en entrée des web services déjà découvert précédemment. Si cela était le cas, cela signifierait que le graphe solution deviendrait cyclique. En conséquence, le programme en tiendra compte en n'utilisant pas ce web service comme nouveau point de départ.

De plus, un mécanisme de pile existe afin de pouvoir « stocker », *n* web services à chaque étape du workflow. Ce système fait que l'ensemble des possibilités d'ordonnancement du workflow seront explorés complètement.

Chaque nouveau web service découvert sera stocké dans un fichier XML qui représente le graphe solution. Ce fichier sera utilisé pour réaliser une représentation graphique du workflow.

Extrait de la classe Workflow. Extrait de la méthode processMatching qui se charge d'effectuer l'intermédiation entre web services.

```
//si score matching suffisant alors web service ok
if (propMatch >= getPropertyScore()) {
        //paramètre sortie
        Output output = (Output) lprofile.getOutputs().get(0);
        float globalScore = computeScore(ontMatch, 0, propMatch);
        Node node = addNewServiceToProcess(parentNode.getOwnerDocument(),
                                           parentNode,
                                           globalScore,
                                           infoWebService.getServiceKey(),
                                           infoWebService.getWsName(),
                                           lprofile,
                                           output.getType().toString());

        //test si on tombe dans un graphe cyclique
        if (! isCyclic(output.getType().toString(),
                  document.getDocumentElement().getFirstChild())) {
                //empile
                stack.push(new StoreInfo(output.getType().toString(),
                infoWebService.getServiceKey(), infoWebService.getWsName(), node));
                } else {
                        //ajoute info cyclique dans xml
                        Element cEl = document.createElement("Cyclic");
                        cEl.setAttribute("ServiceKey", infoWebService.getServiceKey());
                        node.appendChild(cEl);
                        }
                result = true;
```

```
            break;
        }
```

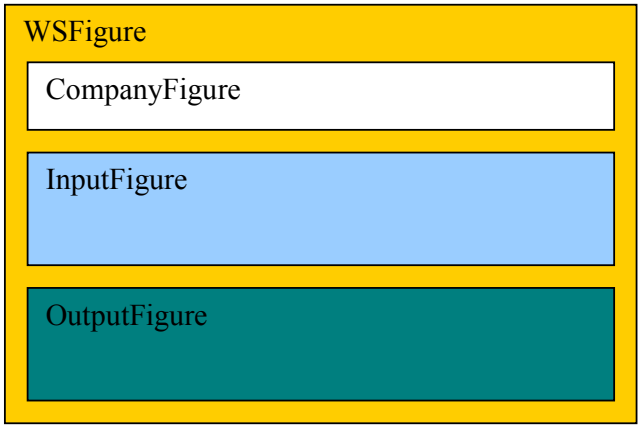
Représentation du workflow

La représentation graphique du workflow découvert se fait à partir d'un fichier XML constitué au fur et à mesure de la découverte dynamique. Ce fichier sera utilisé pour dessiner le graphe solution. Ce graphe orienté est dessiné en s'appuyant sur l'api draw2D qui est un plugin faisant partie de GEF (Graphical Editing Framework) et fournit par IBM dans le cadre du projet éclipse.
Chaque web service est représenté par un bloc et est relié au web service suivant par une flèche. Chaque bloc est composé des sous éléments suivant :

- Le nom du web service ainsi que le score réalisé lors de l'inférence.
- Le nom de l'entreprise proposant le web service.
- La liste des paramètres en entrée.
- La liste des paramètres en sortie.

Au niveau de draw2D chaque bloc ainsi que l'ensemble de ses sous-éléments est définie par un héritage de la classe Figure.

Figure 23 – Représentation d'un web service



Extrait de la classe DrawFlow. Méthode buildNodeFigure qui dessine la représentation graphique d'un web service.

```
/**
 * Dessine une représentation graphique(Figure) d'un Web Service
 *
 * @param contents Gère la liste des objets
 * @param node Node correspondant à la figure que l'on veut dessiner
 * @param contentsLayout XYLayout
 */
private void buildNodeFigure(IFigure contents, org.eclipse.draw2d.graph.Node node,
XYLayout contentsLayout) {
        DataObject dataObject = (DataObject) node.data;

        Label label = new Label(dataObject.getWsName() + " (" +
        dataObject.getMatchingScore() + ")");
        label.setFont(classFont);
        label.setBackgroundColor(ColorConstants.orange);
        label.setOpaque(true);
        label.setBorder(new LineBorder());

        WSFigure wsFigure = new WSFigure(label);
        new Dragger(wsFigure);

        Label lblCompany = new Label(dataObject.getWsCompanyName());
        wsFigure.getCompanyCompartment().add(lblCompany);
```

```
        for (int i =0; i< dataObject.getInputP().size(); i++) {
                Label lblInputP = new Label((String)
                dataObject.getInputP().elementAt(i),
                new Image(getShell().getDisplay(),
                DrawWorkflow.class.getResourceAsStream("arrowr.gif")));
                wsFigure.getInputCompartment().add(lblInputP);
        }

        for (int i =0; i<dataObject.getOutputP().size(); i++) {
                Label lblOutputP = new Label((String)
                dataObject.getOutputP().elementAt(i),
                new Image(getShell().getDisplay(),
                DrawWorkflow.class.getResourceAsStream("arrowl.gif")));
                lblOutputP.setLabelAlignment(PositionConstants.RIGHT);
                wsFigure.getOutputCompartment().add(lblOutputP);
        }

        contents.add(wsFigure);
        contents.getLayoutManager().setConstraint(wsFigure, new Rectangle(node.x,
        node.y, node.width, node.height));
        node.data = wsFigure;
}
```

Figure 24 – Capture d'écran du programme montrant l'affichage graphique du résultat de la découverte dynamique du workflow.

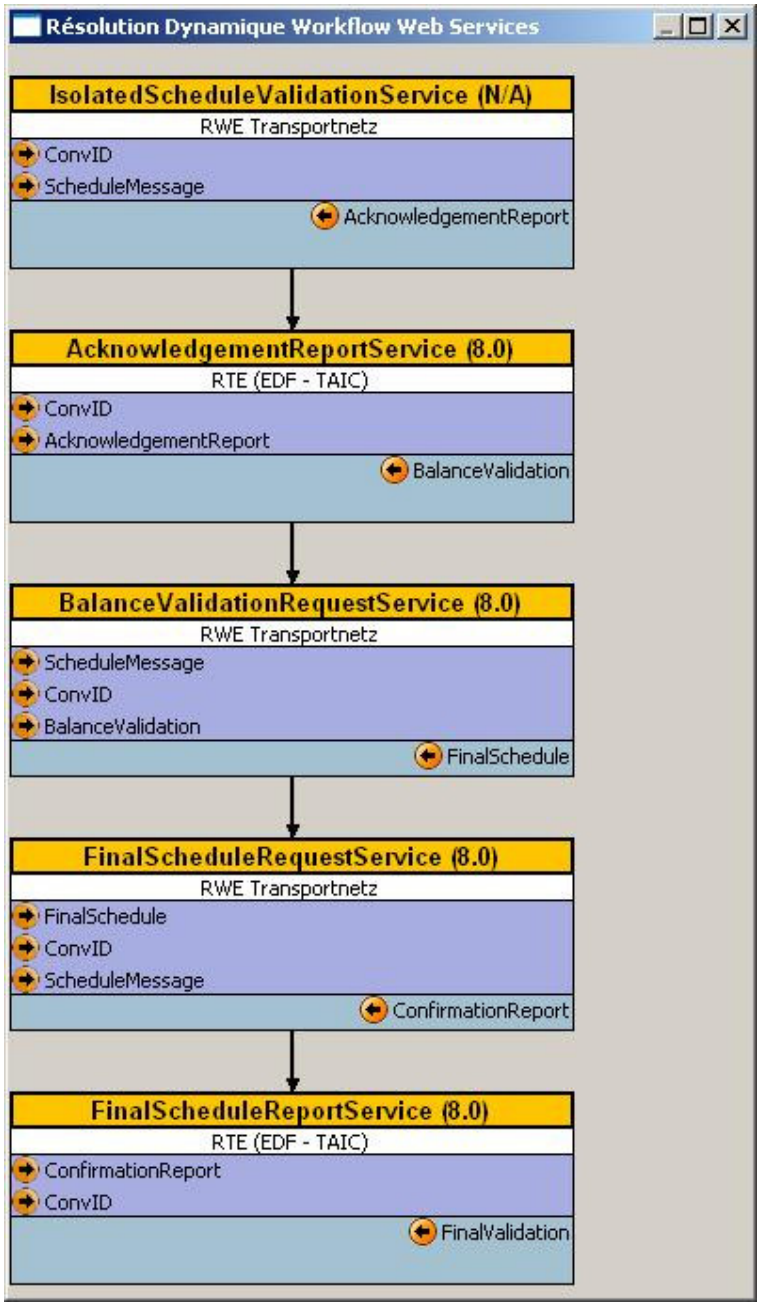


Figure 25 – Diagramme de Classes du package matcher.gui

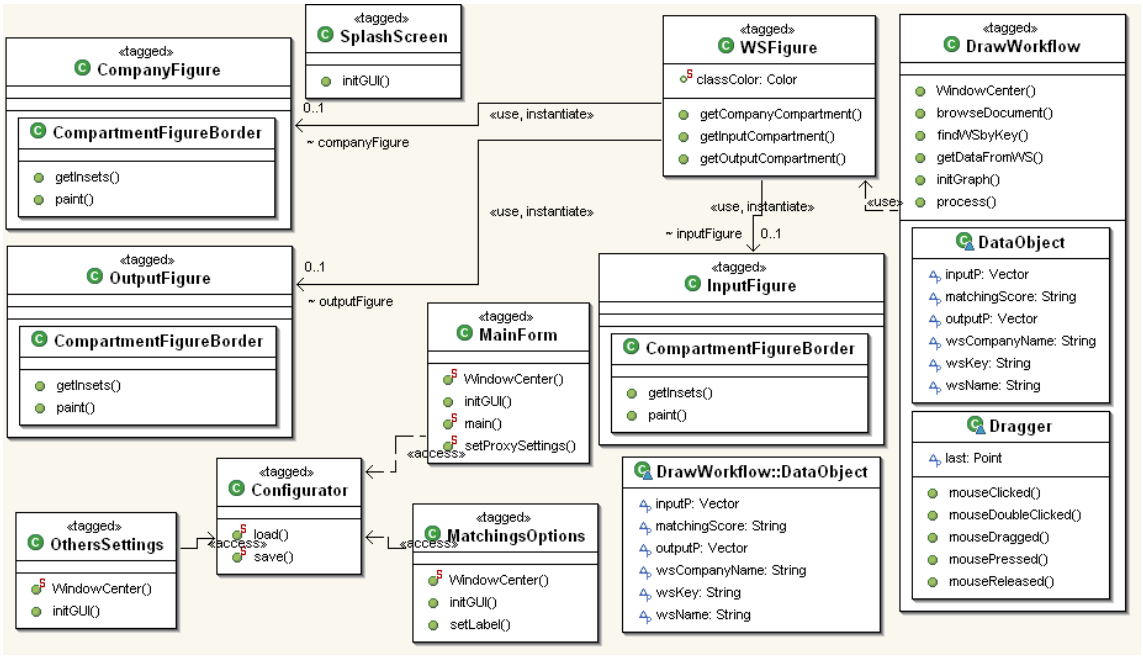
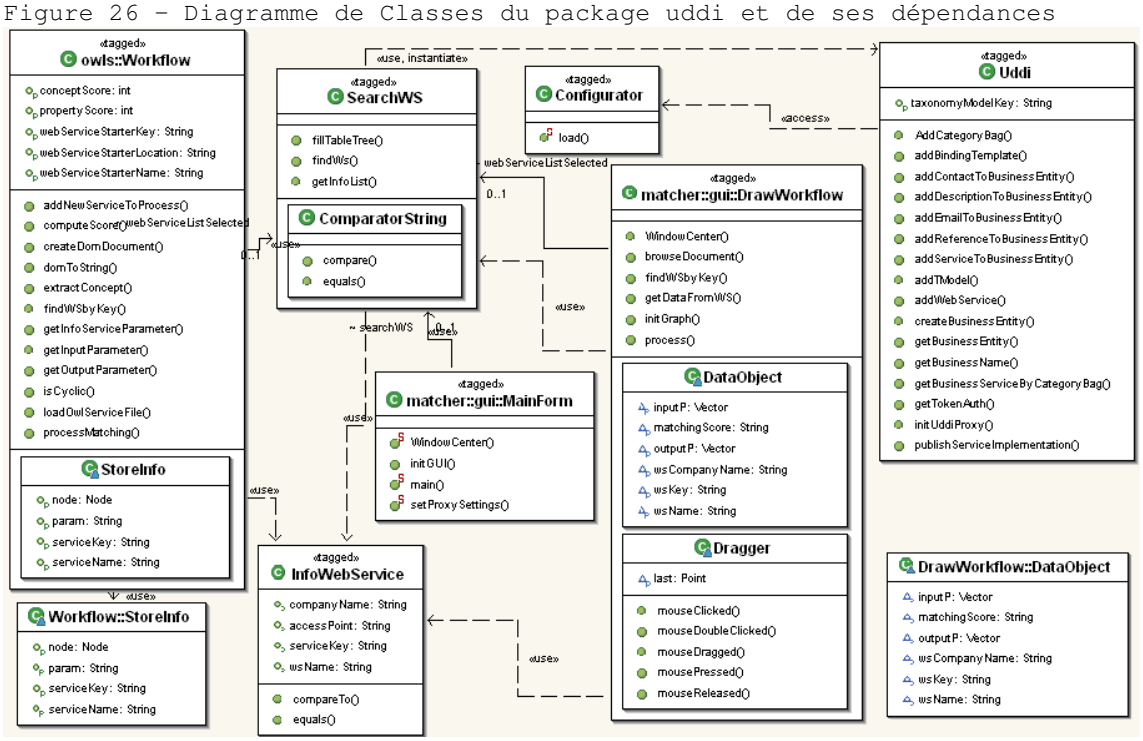


Figure 26 – Diagramme de Classes du package uddi et de ses dépendances



Conclusions

L'utilisation des technologies proposées dans le cadre du Web Sémantique par le W3C a permis de mettre en évidence la possibilité de pouvoir mener à terme la conception et la réalisation d'un prototype viable qui autorise une intermédiation effective entre web services en se basant sur leurs annotations sémantique.

Que ce soit au niveau de la conception des ontologies ou bien des moteurs d'inférences sans oublier les apis permettant de manipuler des données à caractère sémantique, il apparaît qu'un ensemble d'outils est maintenant accessible et permet d'envisager le développement d'applications situées dans le périmètre du Web Sémantique.

Cependant, tous ces outils n'ont pas encore le même niveau de maturité et le degré d'homogénéisation nécessaire ne sera pas atteint avant sans doute plusieurs trimestres. Il apparaît que les outils liés à la conception des ontologies sont les plus matures, ce qui est sans doute à la fois dans l'ordre des choses et le plus significatif.

Au niveau du prototype, la modélisation des contraintes avec SWRL (Semantic Web Rule Language), qui est un langage de règles basé sur RuleML, aurait sans doute apporté une granularité supérieure quand à la détermination de l'intermédiation possible entre web services. Cependant, les outils permettant d'exploiter ces règles sont encore balbutiants et difficilement exploitables. Il apparaît cependant que l'exploitation d'un langage de règles aurait fourni une souplesse que la mesure seule de distance entre les concepts n'est pas en mesure de délivrer.

Concernant l'architecture du prototype, je crois que l'utilisation d'un serveur UDDI s'est avéré intéressante dans l'optique d'une utilisation dans des conditions réelles. En effet, posséder un ou plusieurs annuaires référençant des web services s'est révélé indispensable pour structurer le workflow du processus de découverte.

Le principal manque au cours du développement s'est fait sentir au niveau de l'annotation sémantique des web services, autrement dit, du développement des fichiers OWL-S. Cette phase actuellement sans possibilité d'automatisation est un réel obstacle à la mise en œuvre de ce type de projets. On peut souhaiter que des projets comme Assam Annotator connaissent une réussite critique et permettent d'automatiser ou en tous cas de grandement faciliter l'annotation sémantique en s'appuyant sur des technologies découlant de l'Intelligence Artificielle, comme par exemple, les techniques liées à l'apprentissage par renforcement.

L'utilisation des technologies liées au Web Sémantique se sont donc révélées une voie très prometteuse et le marquage et l'enrichissement sémantique devrait petit à petit gagner une vaste audience et s'immiscer dans une vaste gamme d'applications.

Pour cela toutefois, il faudra que le développement d'ontologies modélisant de vaste domaines de connaissances soit effectué et accessible au plus grand nombre, à travers peut-être le développement d'annuaires publiques. Le Web Sémantique ne connaîtra un développement réel et pérenne qu'à ce prix.

Annexes

UDDI

```
Fichier de configuration pour le paramétrage UDDI4J LocalHost EDF Web
Sémantique

# ------------------------------------------------------------------------
# inquiryURL: The URL for the inquiry API of the target UDDI registry
# publishURL: URL for the publish API of the target UDDI registry
# A list of UDDI URLs is on the UDDI4J website http://www.uddi4j.org/
#
# A typical entry would be of the form
# inquiryURL=http://company.com/uddi_node
# publishURL=https://company.com/uddi_publish_node
# ------------------------------------------------------------------------
# here UDDI test site
inquiryURL = http://localhost:8080/juddi/inquiry
publishURL = http://localhost:8080/juddi/publish

# ------------------------------------------------------------------------
# Userid to use when running the publish samples. Userid/passwords should
# not generally be stored in clear text
# ------------------------------------------------------------------------
#userid = sucrerat
#password = x1fqt92w7aedqc
userid = jdoe
password = password
# ------------------------------------------------------------------------
# Transport classname. Typically defined on commandline as
# -Dorg.uddi4j.TransportClassName=xxx.
# ------------------------------------------------------------------------
# TransportClassName=org.uddi4j.transport.ApacheSOAPTransport
  TransportClassName=org.uddi4j.transport.ApacheAxisTransport
# TransportClassName=org.uddi4j.transport.HPSOAPTransport

# ------------------------------------------------------------------------
# Debug log enabled or not. Typically defined on command line as
# -Dorg.uddi4j.logEnabled=true
# ------------------------------------------------------------------------
logEnabled=false
# logEnabled=true

# ------------------------------------------------------------------------
# Values used to determine the implementation of JSSE to use. Provided
# for convenience, this is typically configured within the jdk
# in JAVA_HOME\jre\lib\security
# ------------------------------------------------------------------------
# Sun JSSE implementation
handlerPackageName=com.sun.net.ssl.internal.www.protocol
securityClassName=com.sun.net.ssl.internal.ssl.Provider

# IBM JSSE implementation
# handlerPackageName=com.ibm.net.ssl.internal.www.protocol
# securityClassName=com.ibm.jsse.JSSEProvider

# ------------------------------------------------------------------------
# UDDI names to use within samples. Samples may or maynot use these values,
# Check the source for the sample
# ------------------------------------------------------------------------
```

```
assertionRelationship=peer-peer

# ---------------------------------------------------------------------
# Additional values can be added as needed as a convenient repository
# for data relevant to the UDDI4J samples
# ---------------------------------------------------------------------


ImbalanceReport-Impl.wsdl

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceReport"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"

xmlns:impl="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceReport-impl"

xmlns:intf="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceReport-intf"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <wsdl:import namespace="http://edf.web.semantique/ImbalanceReport-interface"

      location="http://127.0.0.1:8080/edf.web.semantique/files/uddi/wsdl/ImbalanceReport-Interface.wsdl"/>

      <wsdl:service name="ImbalanceReportService">
            <wsdl:port binding="impl:ImbalanceReportSoapBinding" name="ImbalanceReportPort">
                  <wsdlsoap:address location="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceReport"/>
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>



ImbalanceReport-Interface.wsdl

<wsdl:definitions name="ImbalanceReport-interface"
targetNamespace="http://edf.web.semantique/services/ImbalanceReport-interface"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceReport-impl"
xmlns:intf="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceReport-intf"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
      <wsdl:message name="sendRequest">
            <wsdl:part name="scheduleMessage" type="soapenc:string"/>
            <wsdl:part name="anomalyReport" type="soapenc:string"/>
            <wsdl:part name="convId" type="soapenc:string"/>
      </wsdl:message>
      <wsdl:message name="sendResponse">
            <wsdl:part name="sendReturn" type="soapenc:string"/>
      </wsdl:message>
      <wsdl:portType name="ImbalanceReport">
      <wsdl:operation name="send" parameterOrder="scheduleMessage
anomalyReport convId">
                  <wsdl:input message="impl:sendRequest"
name="sendRequest"/>
                  <wsdl:output message="impl:sendResponse"
name="sendResponse"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="ImbalanceReportSoapBinding"
type="impl:ImbalanceReport">
            <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="send">
                  <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="sendRequest">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://webservices.semantique.web.edf" use="encoded"/>
                  </wsdl:input>
      <wsdl:output name="sendResponse">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/edf.web.semantique/services/ImbalanceRepor
t" use="encoded"/>
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
</wsdl:definitions>
```

Plateforme Web

Fichier web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

      <display-name>ServletController</display-name>
      <servlet>
            <servlet-name>Controller</servlet-name>
            <servlet-class>serveur.Controller</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>Controller</servlet-name>
            <url-pattern>/index</url-pattern>
      </servlet-mapping>

      <display-name>ServletUploadFile</display-name>
      <servlet>
```

```
            <servlet-name>UploadFile</servlet-name>
            <servlet-class>serveur.UploadFile</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>UploadFile</servlet-name>
            <url-pattern>/uploadfile</url-pattern>
      </servlet-mapping>


      <!-- axis part -->
      <display-name>AxisEnabledApp</display-name>
      <servlet>
      <servlet-name>AxisServlet</servlet-name>
      <display-name>Apache-Axis Servlet</display-name>
      <servlet-class>
            org.apache.axis.transport.http.AxisServlet
      </servlet-class>
      </servlet>

      <servlet-mapping>
      <servlet-name>AxisServlet</servlet-name>
      <url-pattern>/AxisServlet</url-pattern>
      </servlet-mapping>


      <servlet-mapping>
      <servlet-name>AxisServlet</servlet-name>
      <url-pattern>/services/*</url-pattern>
      </servlet-mapping>

</web-app>
```

Fichier server-config.wsdd

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultClientConfig"
            xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"

xmlns:handler="http://xml.apache.org/axis/wsdd/providers/handler">

 <globalConfiguration>
   <requestFlow>
     <handler type="java:org.apache.axis.handlers.JWSHandler">
        <parameter name="scope" value="session"/>
     </handler>
     <handler type="java:org.apache.axis.handlers.JWSHandler">
        <parameter name="scope" value="request"/>
        <parameter name="extension" value=".jwr"/>
     </handler>
     <!-- uncomment this if you want the SOAP monitor -->
     <!--
     <handler type="java:org.apache.axis.handlers.SOAPMonitorHandler"/>
     -->
   </requestFlow>
   <!-- uncomment this if you want the SOAP monitor -->
   <!--
   <responseFlow>
     <handler type="java:org.apache.axis.handlers.SOAPMonitorHandler"/>
   </responseFlow>
   -->
 </globalConfiguration>

 <handler type="java:org.apache.axis.handlers.http.URLMapper"
name="URLMapper"/>
```

```
 <handler type="java:org.apache.axis.transport.local.LocalResponder"
name="LocalResponder"/>
 <handler type="java:org.apache.axis.handlers.SimpleAuthenticationHandler"
name="Authenticate"/>

 <service name="AdminService" provider="java:MSG">
  <namespace>http://xml.apache.org/axis/wsdd/</namespace>
  <parameter name="allowedMethods" value="AdminService"/>
  <parameter name="enableRemoteAdmin" value="false"/>
  <parameter name="className" value="org.apache.axis.utils.Admin"/>
 </service>

 <service name="Version" provider="java:RPC">
  <parameter name="allowedMethods" value="getVersion"/>
  <parameter name="className" value="org.apache.axis.Version"/>
 </service>

 <service name="AcknowledgementReport" provider="java:RPC">
  <parameter name="allowedMethods" value="sendAcknowledgement"/>
  <parameter name="className"
value="edf.web.semantique.webservices.AcknowledgementReport"/>
  <parameter name="OWLSNAME" value="AcknowledgementReport.owls"/>
  <transport name="http">
      <parameter name="useDefaultQueryStrings" value="false" />
      <parameter name="qs.owls"
value="edf.web.semantique.axis.QSOwlsHandler"/>
  </transport>
 </service>

 <service name="CorrectedScheduleRequest" provider="java:RPC">
  <parameter name="allowedMethods" value="getNewSchedule"/>
  <parameter name="className"
value="edf.web.semantique.webservices.CorrectedScheduleRequest"/>
 </service>

 <service name="FinalScheduleReport" provider="java:RPC">
  <parameter name="allowedMethods" value="send"/>
  <parameter name="className"
value="edf.web.semantique.webservices.FinalScheduleReport"/>
 </service>

 <service name="ImbalanceReport" provider="java:RPC">
  <parameter name="allowedMethods" value="send"/>
  <parameter name="className"
value="edf.web.semantique.webservices.ImbalanceReport"/>
 </service>

 <service name="BalanceValidationRequest" provider="java:RPC">
  <parameter name="allowedMethods" value="getAnomalyReport"/>
  <parameter name="className"
value="edf.web.semantique.webservices.BalanceValidationRequest"/>
 </service>

 <service name="FinalScheduleRequest" provider="java:RPC">
  <parameter name="allowedMethods" value="getFinalSchedule"/>
  <parameter name="className"
value="edf.web.semantique.webservices.FinalScheduleRequest"/>
 </service>

 <service name="IsolatedScheduleValidation" provider="java:RPC">
  <parameter name="allowedMethods" value="send"/>
  <parameter name="className"
value="edf.web.semantique.webservices.IsolatedScheduleValidation"/>
 </service>

 <transport name="http">
  <parameter name="qs.owls" value="edf.web.semantique.axis.QSOwlsHandler"/>

  <requestFlow>
   <handler type="URLMapper"/>
```

```
   <handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
  </requestFlow>
 </transport>

 <transport name="local">
  <responseFlow>
   <handler type="LocalResponder"/>
  </responseFlow>
 </transport>

</deployment>
```

Javadoc

---

owls
Class Workflow

```
java.lang.Object
  └owls.Workflow
```

---

public class **Workflow**
extends java.lang.Object

Résout un workflow type bpel dynamiquement en utilisant des ontologies et chaque fichier OWL-S associé à chaque Web Service

Version:
1.0
Author:
herve perez

Constructor Summary

Workflow(SearchWS webServiceListS, java.lang.String webServiceStarterK, int conceptScore, int parameterScore, org.eclipse.swt.widgets.Text text, org.eclipse.swt.widgets.ProgressBar progressBar)

Method Summary

org.w3c.dom.Node	**addNewServiceToProcess**(org.w3c.dom.Document doc, org.w3c.dom.Node parentNode, float globalScore, java.lang.String ServiceKey, java.lang.String WebSe org.mindswap.owls.profile.Profile lprofile, java.lang.String outputParam)

	Ajoute un nouveau web service au process
float	**computeScore**(int ontMatch, int propMatchInput, int propMatchOutput) Calcule la note finale du matching
org.w3c.dom.Document	**createDomDocument**(java.lang.String rootNodeName, java.lang.String serviceKey, java.lang.String Servi org.mindswap.owls.profile.Profile profile, java.lang.String outputParam) Création d'un Document DOM qui stockera le graphe sol
java.lang.String	**domToString**(org.w3c.dom.Element element) Transforme un arbre DOM en string.
java.lang.String	**extractConcept**(java.lang.String delimiter, java.lang.String uriConcept) Extrait le Nom qualifé d'une URI
InfoWebService	**findWSbyKey**(java.lang.String serviceKey) Retourne une instance de InfoWebService
int	**getConceptScore**()
org.w3c.dom.Document	**getDocument**()
java.lang.String	**getInfoServiceParameter**(java.lang.String qName, org.mindswap.owls.profile.Profile lprofile) Parcours le fichier OWL-S avec API Jena pour recherche inforamtions
java.lang.String[]	**getInputParameter**(org.mindswap.owls.profile.Profile Récupère les paramètres en entrée
java.lang.String	**getOutputParameter**(org.mindswap.owls.profile.Profil Récupère le paramètre en sortie
org.eclipse.swt.widgets.ProgressBar	**getProgressBarReport**()
int	**getPropertyScore**()
org.eclipse.swt.widgets.Text	**getTextReport**()
SearchWS	**getWebServiceListSelected**()
java.lang.String	**getWebServiceStarterKey**()
java.lang.String	**getWebServiceStarterLocation**()
java.lang.String	**getWebServiceStarterName**()
boolean	**isCyclic**(java.lang.String inputParam, org.w3c.dom.N Détermine si le graphe va devenir cyclique
void	**loadOwlServiceFile**(java.lang.String uriFile) Load le Service File et le profile

boolean	**processMatching**(java.lang.String classMaster, java.lang.String ouputParam, java.lang.String servi org.w3c.dom.Node parentNode) Parse l'ensemble des Web Services pour détecter si un ou Web Services sont "compatibles avec le précédent
void	**setConceptScore**(int i)
void	**setDocument**(org.w3c.dom.Document document)
void	**setProgressBarReport**(org.eclipse.swt.widgets.Progre
void	**setPropertyScore**(int i)
void	**setTextReport**(org.eclipse.swt.widgets.Text text)
void	**setWebServiceListSelected**(SearchWS searchWS)
void	**setWebServiceStarterKey**(java.lang.String string)
void	**setWebServiceStarterLocation**(java.lang.String strir
void	**setWebServiceStarterName**(java.lang.String string)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Workflow

```
public Workflow(SearchWS webServiceListS,
                java.lang.String webServiceStarterK,
                int conceptScore,
                int parameterScore,
                org.eclipse.swt.widgets.Text text,
                org.eclipse.swt.widgets.ProgressBar progressBar)
```

Parameters:

`webServiceListS` - instance de SearchWS référencant l'ensemble de la liste des Web Services proposés par le Serveur UDDI

`webServiceStarterK` - Service Key du web service de départ

`conceptScore` - Score Minimun de Matching pour les Concepts

`parameterScore` - score Minimun de Matching pour les paramètres

`text` - Widget SWT Text permettant de suivre l'évolution du process de découverte dynamique

`progressBar` - Widget SWT Text permettant de suivre l'évolution du process de découverte dynamique

Method Detail

processMatching

```
public boolean processMatching(java.lang.String classMaster,
                               java.lang.String ouputParam,
                               java.lang.String serviceKey,
                               org.w3c.dom.Node parentNode)
```

Parse l'ensemble des Web Services pour détecter si un ou plusieurs Web Services sont "compatibles avec le précédent

Parameters:

`classMaster` - Concept Référence servant au matching pour les concepts

`ouputParam` - Paramètre Output servant au matching pour les paramètres

`serviceKey` - Service Key du Web Service parent

`parentNode` - Node du document DOM représentant le web service parent

Returns:

boolean

---

findWSbyKey

```
public InfoWebService findWSbyKey(java.lang.String serviceKey)
```

Retourne une instance de InfoWebService

Parameters:

`serviceKey` - Service Key du Web Service

Returns:

InfoWebService Retourne une instance InfoWebService correspondant au Web Service

---

loadOwlServiceFile

```
public void loadOwlServiceFile(java.lang.String uriFile)
```

Load le Service File et le profile

Parameters:

`uriFile` - URI du fichier OWL-S

---

getInputParameter

```
public java.lang.String[]
getInputParameter(org.mindswap.owls.profile.Profile lprofile)
```

Récupère les paramètres en entrée

Parameters:

`lprofile` - Profile du fichier OWL-S

Returns:

String[] Liste des paramètres en entrée

---

getOutputParameter

```
public java.lang.String
getOutputParameter(org.mindswap.owls.profile.Profile lprofile)
```

Récupère le paramètre en sortie

Parameters:

`lprofile` - Profile du fichier OWL-S

Returns:

String Paramètre de Sortie

---

getInfoServiceParameter

```
public java.lang.String getInfoServiceParameter(java.lang.String qName,

org.mindswap.owls.profile.Profile lprofile)
```

Parcours le fichier OWL-S avec API Jena pour rechercher des inforamtions

Parameters:

`qName` - Qualified Name que l'on recherche

`lprofile` - Profile du fichier OWL-S

Returns:

String Information recherchée

---

computeScore

```
public float computeScore(int ontMatch,
                          int propMatchInput,
                          int propMatchOutput)
```

Calcule la note finale du matching

Parameters:

`ontMatch` - obtenue sur le Matching sur les concepts

`propMatchInput` - obtenue sur le Matching sur des paramètres en entrée

`propMatchOutput` - obtenue sur le Matching sur des paramètres en sortie

Returns:

float Note finale du matching

---

addNewServiceToProcess

```
public org.w3c.dom.Node addNewServiceToProcess(org.w3c.dom.Document doc,
                                               org.w3c.dom.Node parentNode,
                                               float globalScore,
                                               java.lang.String ServiceKey,

java.lang.String WebServiceName,

org.mindswap.owls.profile.Profile lprofile,

java.lang.String outputParam)
```

Ajoute un nouveau web service au process

Returns:

Node

---

isCyclic

```
public boolean isCyclic(java.lang.String inputParam,
                        org.w3c.dom.Node node)
```

Détermine si le graphe va devenir cyclique

Parameters:

`inputParam` - Paramètre en entrée

`node` - Node correpsondant au Web Service de départ

Returns:
boolean

---

createDomDocument

```
public org.w3c.dom.Document
createDomDocument(java.lang.String rootNodeName,
                                          java.lang.String serviceKey,
                                          java.lang.String ServiceName,

org.mindswap.owls.profile.Profile profile,
                                          java.lang.String outputParam)
```

Création d'un Document DOM qui stockera le graphe solution
Returns:
Document

---

extractConcept

```
public java.lang.String extractConcept(java.lang.String delimiter,
                                       java.lang.String uriConcept)
```

Extrait le Nom qualifé d'une URI
Parameters:
`delimiter` - délimiteur
`uriConcept` - URI
Returns:
String

---

domToString

```
public java.lang.String domToString(org.w3c.dom.Element element)
```

Transforme un arbre DOM en string. Utilisé en mode debug
Parameters:
`element` - Element
Returns:
String

---

getWebServiceStarterLocation

```
public java.lang.String getWebServiceStarterLocation()
```

Returns:
String

---

getWebServiceStarterName

```
public java.lang.String getWebServiceStarterName()
```

Returns:
String

---

setWebServiceStarterLocation

```
public void setWebServiceStarterLocation(java.lang.String string)
```

Parameters:
`string` -

---

setWebServiceStarterName

`public void` **`setWebServiceStarterName`**`(java.lang.String string)`

Parameters:
`string` -

---

getWebServiceListSelected

`public` SearchWS **`getWebServiceListSelected`**`()`

Returns:
SearchWS

---

setWebServiceListSelected

`public void` **`setWebServiceListSelected`**`(`SearchWS `searchWS)`

Parameters:
`searchWS` -

---

getWebServiceStarterKey

`public java.lang.String` **`getWebServiceStarterKey`**`()`

Returns:
String

---

setWebServiceStarterKey

`public void` **`setWebServiceStarterKey`**`(java.lang.String string)`

Parameters:
`string` -

---

getConceptScore

`public int` **`getConceptScore`**`()`

Returns:
int

---

getPropertyScore

`public int` **`getPropertyScore`**`()`

Returns:
int

---

setConceptScore

`public void` **`setConceptScore`**`(int i)`

Parameters:
`i` -

setPropertyScore

```
public void setPropertyScore(int i)
```

Parameters:

i -

---

getTextReport

```
public org.eclipse.swt.widgets.Text getTextReport()
```

Returns:

Text

---

setTextReport

```
public void setTextReport(org.eclipse.swt.widgets.Text text)
```

Parameters:

text -

---

getDocument

```
public org.w3c.dom.Document getDocument()
```

Returns:

Document

---

setDocument

```
public void setDocument(org.w3c.dom.Document document)
```

Parameters:

document -

---

getProgressBarReport

```
public org.eclipse.swt.widgets.ProgressBar getProgressBarReport()
```

Returns:

ProgressBar

---

setProgressBarReport

```
public void setProgressBarReport(org.eclipse.swt.widgets.ProgressBar bar)
```

Parameters:

bar -

---

matcher.gui
Class DrawWorkflow

```
java.lang.Object
  └org.eclipse.swt.widgets.Widget
      └org.eclipse.swt.widgets.Control
          └org.eclipse.swt.widgets.Scrollable
              └org.eclipse.swt.widgets.Composite
                  └matcher.gui.DrawWorkflow
```

All Implemented Interfaces:

org.eclipse.swt.graphics.Drawable

public class **DrawWorkflow**
extends org.eclipse.swt.widgets.Composite

Classe gérant l'affichage du graphe solution

Version:

1.0

Author:

herve perez

Field Summary

Fields inherited from class org.eclipse.swt.widgets.Control
handle

Constructor Summary

DrawWorkflow(org.eclipse.swt.widgets.Composite parent, int style)

Method Summary

void	**browseDocument**(org.w3c.dom.Document document, org.w3c.dom.Node nodeDOM, org.eclipse.draw2d.graph.Node nodeP) Parcours récursivement l'arbre DOM du workflow et construit la représentation graphique
InfoWebService	**findWSbyKey**(java.lang.String serviceKey) Retourne l'instance de InfoWebService correspond au serviceKey

matcher.gui.DrawWorkflow.DataObject	**getDataFromWS**(org.w3c.dom.Node node, java.lang.String wsKey) Récupère l'ensemble des données nécessaires à la représentation graphique d'un node
SearchWS	**getWebServiceListSelected**()
org.w3c.dom.Document	**getWorkflow**()
void	**initGraph**() Initialise le graphe draw2d
void	**process**() Gestion du process rendu graphique
void	**setWebServiceListSelected**(SearchWS searchWS)
void	**setWorkflow**(org.w3c.dom.Document document)
void	**WindowCenter**(org.eclipse.swt.widgets.Display displa org.eclipse.swt.widgets.Shell shell) Centre la fenêtre d'affichage

Methods inherited from class org.eclipse.swt.widgets.Composite
computeSize, getChildren, getLayout, getTabList, layout, layout, setFocus, setLayout, setTabList

Methods inherited from class org.eclipse.swt.widgets.Scrollable
computeTrim, getClientArea, getHorizontalBar, getVerticalBar

Methods inherited from class org.eclipse.swt.widgets.Control
addControlListener, addFocusListener, addHelpListener, addKeyListener, addMouseListener, addMouseMoveListener, addMouseTrackListener, addPaintListener, addTraverseListener, computeSize, forceFocus, getAccessible, getBackground, getBorderWidth, getBounds, getDisplay, getEnabled, getFont, getForeground, getLayoutData, getLocation, getMenu, getParent, getShell, getSize, getToolTipText, getVisible, internal_dispose_GC, internal_new_GC, isDisposed, isEnabled, isFocusControl, isReparentable, isVisible, moveAbove, moveBelow, pack, pack, redraw, redraw, removeControlListener, removeFocusListener, removeHelpListener, removeKeyListener, removeMouseListener, removeMouseMoveListener, removeMouseTrackListener, removePaintListener, removeTraverseListener, setBackground, setBounds, setBounds, setCapture, setCursor, setEnabled, setFont, setForeground, setLayoutData, setLocation, setLocation, setMenu, setParent, setRedraw, setSize, setSize, setToolTipText, setVisible, toControl, toControl, toDisplay, toDisplay, traverse, update

Methods inherited from class org.eclipse.swt.widgets.Widget
addDisposeListener, addListener, dispose, getData, getData, getStyle, notifyListeners, removeDisposeListener, removeListener, setData, setData,

toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

DrawWorkflow

```
public DrawWorkflow(org.eclipse.swt.widgets.Composite parent,
                    int style)
```

Parameters:

`parent` -

`style` - Style fenêtre

Method Detail

process

```
public void process()
```

Gestion du process rendu graphique

---

initGraph

```
public void initGraph()
```

Initialise le graphe draw2d

---

browseDocument

```
public void browseDocument(org.w3c.dom.Document document,
                           org.w3c.dom.Node nodeDOM,
                           org.eclipse.draw2d.graph.Node nodeP)
```

Parcours récursivement l'arbre DOM du workflow et construit la représentation graphique

Parameters:

`document` - Document DOM

`nodeDOM` - Node DOM

`nodeP` - Node draw2d

---

getDataFromWS

```
public matcher.gui.DrawWorkflow.DataObject
getDataFromWS(org.w3c.dom.Node node,

java.lang.String wsKey)
```

Récupère l'ensemble des données nécessaires à la représentation graphique d'un node

Parameters:

`node` - Node DOM

`wsKey` - String représentant le ServiceKey UDDI correspondant au Web Service

Returns:

DataObject Retourne une instance de DataObject

findWSbyKey

```
public InfoWebService findWSbyKey(java.lang.String serviceKey)
```

Retourne l'instance de InfoWebService correspond au serviceKey

Parameters:

`serviceKey` - String représentant le ServiceKey UDDI correspondant au Web Service

Returns:

InfoWebService Retourne l'instance InfoWebService correspond au serviceKey

---

WindowCenter

```
public void WindowCenter(org.eclipse.swt.widgets.Display display,
                         org.eclipse.swt.widgets.Shell shell)
```

Centre la fenêtre d'affichage

Parameters:

`display` - Display SWT

`shell` - Shell SWT

---

setWebServiceListSelected

```
public void setWebServiceListSelected(SearchWS searchWS)
```

Parameters:

`searchWS` -

---

getWebServiceListSelected

```
public SearchWS getWebServiceListSelected()
```

Returns:

SearchWS

---

getWorkflow

```
public org.w3c.dom.Document getWorkflow()
```

Returns:

Document

---

setWorkflow

```
public void setWorkflow(org.w3c.dom.Document document)
```

Parameters:

`document` -

---

uddi
Class SearchWS

```
java.lang.Object
  └uddi.SearchWS
```

public class **SearchWS**
extends java.lang.Object

Classe s'occupant de la gestion des requêtes UDDI

Version:
1.0
Author:
herve perez

Nested Class Summary

class	**SearchWS.ComparatorString** Comparateur faisant un tri sur le nom des web services

Constructor Summary

SearchWS()

Method Summary

void	**fillTableTree**(org.eclipse.swt.custom.TableTree tableTreeResultSearch) Affiche les Web Services dans un TableTree
void	**findWs**(org.eclipse.swt.custom.TableTree tableTreeResultSearch, java.lang.String categoryBagName, java.lang.String categoryBagValue, int maxRows) Recherche et affiche dans un TableTree une liste de Web Services
java.util.Vector	**getInfoList**(java.lang.String categoryBagName, java.lang.String categoryBagValue, int maxRows) Récupère l'ensemble des Web Services correspondant au categorybag passé en paramètre sur un serveur UDDI
java.util.TreeSet	**getListWsSorted**()
void	**setListWsSorted**(java.util.TreeSet set)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SearchWS

```
public SearchWS()
```

Method Detail

getInfoList

```
public java.util.Vector getInfoList(java.lang.String categoryBagName,
                                    java.lang.String categoryBagValue,
                                    int maxRows)
```

Récupère l'ensemble des Web Services correspondant au categorybag passé en paramètre sur un serveur UDDI

Parameters:

`categoryBagName` - (Nom du categorybag)

`categoryBagValue` - (valeur du categorybag code naics 1997)

`maxRows` - (nb de résultat ramené par le serveur uddi)

Returns:

TreeSet

---

findWs

```
public void findWs(org.eclipse.swt.custom.TableTree tableTreeResultSearch,
                   java.lang.String categoryBagName,
                   java.lang.String categoryBagValue,
                   int maxRows)
```

Recherche et affiche dans un TableTree une liste de Web Services

Parameters:

`tableTreeResultSearch` - (TableTree affichant les résultats)

`categoryBagName` - (Nom du categorybag)

`categoryBagValue` - (valeur du categorybag code naics 1997)

`maxRows` - (nb de résultats ramené par le serveur uddi)

---

fillTableTree

```
public void
fillTableTree(org.eclipse.swt.custom.TableTree tableTreeResultSearch)
```

Affiche les Web Services dans un TableTree

Parameters:

`tableTreeResultSearch` - (TableTree affichant les résultats)

---

getListWsSorted

```
public java.util.TreeSet getListWsSorted()
```

Returns:

TreeSet

---

setListWsSorted

```
public void setListWsSorted(java.util.TreeSet set)
```

Parameters:

set -

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

racer
Class Racer

```
java.lang.Object
  └ racer.Racer
```

---

public class **Racer**
extends java.lang.Object

Classe supervisant la connection au moteur d'inférence Racer et l'inférence

Version:

1.0

Author:

herve perez

Constructor Summary

Racer()	

Method Summary

static RacerClient	**getConnectionToRacer**(java.lang.String urlFile) Connection Racer et chargement de l'ontologie
static int	**matchConcepts**(java.lang.String concept1, java.lang.String concept2, java.lang.String uriOntology) Effectue un matching sur des concepts
static int	**matchProperties**(java.lang.String property1, java.lang.String property2, java.lang.String uriOntology) Effectue un matching sur des roles
static boolean	**testConnectionToRacer**()

	Teste si la connection avec Racer est OK

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Racer

```
public Racer()
```

Method Detail

matchConcepts

```
public static int matchConcepts(java.lang.String concept1,
                                java.lang.String concept2,
                                java.lang.String uriOntology)
```

Effectue un matching sur des concepts

Parameters:

`concept1` -

`concept2` -

`uriOntology` - uri de l'ontologie sur laquelle on effectue l'inférence

Returns:

int correspondant au degrè de satisfaction du matching

---

matchProperties

```
public static int matchProperties(java.lang.String property1,
                                  java.lang.String property2,
                                  java.lang.String uriOntology)
```

Effectue un matching sur des roles

Parameters:

`property1` -

`property2` -

`uriOntology` - uri de l'ontologie sur laquelle on effectue l'inférence

Returns:

int correspondant au degrè de satisfaction du matching

---

getConnectionToRacer

```
public static RacerClient getConnectionToRacer(java.lang.String urlFile)
```

Connection Racer et chargement de l'ontologie

Parameters:

`urlFile` -

Returns:

RacerClient

---

testConnectionToRacer

```
public static boolean testConnectionToRacer()
```

Teste si la connection avec Racer est OK

Returns:

boolean

---

utils
Class ProxyEdf

```
java.lang.Object
  └ java.net.Authenticator
      └ utils.ProxyEdf
```

---

public class **ProxyEdf**
extends java.net.Authenticator

Version:

1.0

Author:

herve perez

Constructor Summary

`ProxyEdf()`

Methods inherited from class java.net.Authenticator
`requestPasswordAuthentication, requestPasswordAuthentication, setDefault`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ProxyEdf

```
public ProxyEdf()
```

Fichier de configuration du programme de découverte dynamique de workflow

Fichier config.prop

```
##Fichier de paramétrages
#Thu Aug 26 16:55:02 CEST 2004
parametersMatching=4
uddiServerUrl=http\://localhost\:8080/juddi/inquiry
addressIPRacer=130.98.172.107
conceptsMatching=3
```

JSP

ontology_list.jsp

```
<%-- h. Perez / JSP administration des ontologies --%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<% String path = "http://" + request.getLocalAddr() + ":" +
request.getLocalPort()+ request.getContextPath();%>

<html>
<head>
      <title>Administration des Ontologies</title>
      <link rel="stylesheet" href="<%=path%>/resources/css/020.css"
type="text/css"/>
      <script language="Javascript"
src="<%=path%>/resources/javascript/edf.js"
type="text/javascript"></script>
</head>
<body>
      <table border="1" WIDTH="100%">
            <tr>
                  <td wrap WIDTH="50%" id="mainmenu">Liste des
Ontologies</td>
                  <td wrap WIDTH="50%">
                        <a href="<%=path%>/jsp/ontology_add.jsp"
target="MainFrame">Ajouter une Ontologie</a>
                  </td>
            </tr>
      </table>

      <jsp:useBean id="BeanOwls" class="serveur.BeanOwls"
scope="request"></jsp:useBean>
      <%=BeanOwls.getOwlsFileList(path)%>

</body>
</html>
```

ontology_add.jsp

```
<%-- h. Perez / JSP administration des ontologies --%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<% String path = "http://" + request.getLocalAddr() + ":" +
request.getLocalPort()+ request.getContextPath();%>

<html>
<head>
      <title>Administration des agents VDL</title>
      <link rel="stylesheet" href="<%=path%>/resources/css/020.css"
type="text/css">
      <script language="Javascript"
src="<%=path%>/resources/javascript/edf.js"
type="text/javascript"></script>
</head>
<body>
      <table border="1" WIDTH="100%">
            <tr>
                  <td wrap WIDTH="50%">
                        <a
href="<%=request.getContextPath()%>/jsp/ontology_list.jsp"
target="MainFrame">Liste des Ontologies</a>
                  </td>
                  <td wrap WIDTH="50%" id="mainmenu">Ajouter une
Ontologie</td>
            </tr>
      </table>

      <form name="AddOntology" ENCTYPE="multipart/form-data"  method="POST"
action="<%=request.getContextPath()%>/uploadfile">
      <table>
            <tr>
                  <th colspan="2">Ajouter une Ontology</th>
            </tr>
            <tr>
                  <td>Nom Ontologie</td>
                  <td><INPUT TYPE="text" NAME="name"></td>
            </tr>
            <tr>
                  <td>Fichier</td>
                  <td><INPUT TYPE="file" NAME="file"></td>
            </tr>
            <tr>
                  <td>Auteur</td>
                  <td><INPUT TYPE="text" NAME="author"></td>

            </tr>
            <tr>
                  <td>Informations</td>
                  <td><textarea NAME="informations" rows="5"
cols="50"></textarea></td>
            </tr>

            <tr>
                  <td></td>
                  <td align="right"><INPUT TYPE="submit" VALUE="Ajouter
Ontologie"></td>
            </tr>
      </table>
</body>
</html>
```

Servlet Controller

```
package serveur;


import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.*;


import utils.*;


/**
 * Servlet faisant office de controller (modèle mvc 2).
 *
 * @author herve perez
 * @version 1.0
 *
 *
*/

public class Controller extends HttpServlet {
      private boolean debugMode = false;

      /**
       * @param req
       * @param res
       */
      protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

            String path = "http://" + req.getLocalAddr() + ":" +
req.getLocalPort()+ req.getContextPath();
            RequestDispatcher      disp = null;
            String                       inParam1 = null;
            String                       inParam2 = null;
            String                       param1 = "page";
            String                       param2 = "action";
            String                       param3 = "idName";

            try {
                  inParam1 = req.getParameter(param1);
                  if (debugMode) System.out.println("param=" + inParam1);

                  if (inParam1 != null) {

                        ///////////////main
                        if (inParam1.compareTo("home") == 0)
                              disp =
req.getRequestDispatcher("/jsp/main.jsp");

                        ///////////////ontology
                        if (inParam1.compareTo("ontology") == 0)
                              disp =
req.getRequestDispatcher("/jsp/ontology_list.jsp");
                  } else {
                        inParam1 = req.getParameter(param2);
                        inParam2 = req.getParameter(param3);
```

```
                        if ((inParam1 != null) && (inParam2 != null)) {

                              ///////////////inspect
                              if (inParam1.compareTo("inspect") == 0) {
                                    res.setContentType("text/xml");
                                    PrintWriter out =
ServletUtils.getHeader(res);
                                    BeanOwls ontology = new BeanOwls();

      out.println(Transform.convert(ontology.getOwlsFileFromDb(inParam2)));
                              }

                              ///////////////informations

                              if (inParam1.compareTo("informations") == 0) {
                                    res.setContentType("text/html");
                                    PrintWriter out =
ServletUtils.getHeaderForHtml(res, path);

                                    BeanOwls ontology = new BeanOwls();
                                    String[] infos =
ontology.getInformationsOntology(inParam2);
                                    out.println("<body>");
                                    for(int i=0; i<infos.length; i++)
out.println(infos[i] + "<br>");
                                    out.println("</body></html>");
                              }

                              ///////////////delete
                              if (inParam1.compareTo("delete") == 0) {
                                    BeanOwls ontology = new BeanOwls();
                                    ontology.deleteOwlsFile(inParam2);
                                    disp =
req.getRequestDispatcher("/jsp/ontology_list.jsp");
                              }

                        } else {
                              disp =
req.getRequestDispatcher("/jsp/index.jsp");

                        }
                  }

                  if (disp != null) disp.forward(req, res);
            }
            catch (Exception e) {
                  if (debugMode) System.out.println(e.getMessage());
                  disp = req.getRequestDispatcher("/jsp/index.jsp");
                  disp.forward(req, res);
            }
      }
}
```

Servlet UploadFile

```
package serveur;


import java.io.*;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.*;
```

```
import org.apache.commons.fileupload.DiskFileUpload;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadBase;
import org.apache.commons.fileupload.FileUploadException;
import java.util.Iterator;
import java.util.List;

import java.sql.*;

import utils.Database;

/**
 *  Servlet qui permet d'uploader un fichier et de le stocker dans la base
de données
 * @author herve perez
 * @version 1.0
 */
public class UploadFile extends HttpServlet {
      private boolean debugMode = false;


      /**
       * @param req
       * @param res
       */
      protected void doPost(HttpServletRequest req, HttpServletResponse
res)
            throws ServletException, IOException {

            RequestDispatcher disp = null;

            insertOwlsFile(req, res);

            //refresh la liste des fichiers owl-s
            disp = req.getRequestDispatcher("/jsp/ontology_list.jsp");
            disp.forward(req, res);
      }


      /**
       * Insère le fichier Owls dans la base
       *
       * @param req
       * @param res
       *
       * @throws ServletException
       * @throws IOException
       */
      private void insertOwlsFile(
            HttpServletRequest req,
            HttpServletResponse res)
            throws ServletException, IOException {
            Statement statement = null;
            ResultSet rs = null;
            String name = null;
            String value = null;
            boolean isMultipart = false;
            PreparedStatement ps = null;
            String s =
                  "insert into ontology(ontology_Name, ontology_Data,
ontology_Author, ontology_Information) values(?, ?, ?, ?)";
            int i = 0;

            Database db = new Database();
            Connection connection = db.getConnection(0);

            PrintWriter out = res.getWriter();
            isMultipart = FileUploadBase.isMultipartContent(req);
            FileUploadBase.isMultipartContent(req);
```

```
            DiskFileUpload upload = new DiskFileUpload();

            try {
                  statement = connection.createStatement();
                  ps = connection.prepareStatement(s);

                  List items = upload.parseRequest(req);
                  Iterator iter = items.iterator();
                  while (iter.hasNext()) {
                        FileItem item = (FileItem) iter.next();

                        if (item.isFormField()) {
                              name = item.getFieldName();
                              value = item.getString();
                              i++;
                              try {
                                    ps.setString(i, value);
                              } catch (SQLException e2) {
                                    e2.printStackTrace();
                              }
                        } else {
                              InputStream fileOnto = item.getInputStream();
                              ps.setBinaryStream(2, fileOnto,
ps.getMaxFieldSize());
                              i = 2;
                        }
                  }

                  if (ps.execute())
                        if (debugMode)
                              System.out.print("owls inséré dans la base");

                  db.disconnect();

            } catch (SQLException e1) {
                  e1.printStackTrace();
            } catch (FileUploadException e) {
                  e.printStackTrace();
            }
      }

}
```

Ontologies OWL

Ontologie ConceptsETSO

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://a.com/ontology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://a.com/ontology">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="AR">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >System Operator</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#ScheduleExchange"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Generation">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Etso"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="FinalScheduleRequest">
```

```
    <rdfs:subClassOf rdf:resource="#AR"/>
  </owl:Class>
  <owl:Class rdf:ID="BalanceValidationRequest">
    <rdfs:subClassOf rdf:resource="#AR"/>
  </owl:Class>
  <owl:Class rdf:ID="CorrectedScheduleRequest">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#AE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Etso">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >
            classe racine de l'ontologie etso (european  transmission
system operator)</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="ImbalanceReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#AE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="FinalScheduleReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#AE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AcknowledgementReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#AE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Forecast">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Prévisions</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Data"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ImportExportBalance">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#TCU"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ScheduleExchange">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Programmation des échanges</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Interconnexion"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Mechanism">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Interconnexion"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >
                Mécanismes d'interconnexions</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="RealTime">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Données en temps réels</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Data"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Data">
    <rdfs:subClassOf rdf:resource="#Etso"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Informations sur le réseau</rdfs:comment>
  </owl:Class>
```

```
  <owl:Class rdf:ID="Interconnexion">
    <rdfs:subClassOf rdf:resource="#Etso"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >
            Données sur les capacités d'interconnexion entre
pays</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="PlannedOutage">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Réduction prévues dans la capacité de connection de transmission entre
pays pour une période couvrant les six prochaines semaines.
            </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Grid"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Capacity">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Capacités d'interconnexions</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Interconnexion"/>
  </owl:Class>
  <owl:Class rdf:ID="TCU">
    <rdfs:subClassOf rdf:resource="#Interconnexion"/>
  </owl:Class>
  <owl:Class rdf:ID="AE">
    <rdfs:subClassOf rdf:resource="#ScheduleExchange"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Balance Responsible Party</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Grid">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Données sur la "grille" des disponibiltés</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Etso"/>
  </owl:Class>
  <owl:Class rdf:ID="IsolatedScheduleValidation">
    <rdfs:subClassOf rdf:resource="#AR"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Schedule Message  Validation</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="SystemDisturbance">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Données sur les pertubations
            </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Grid"/>
  </owl:Class>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.2 beta, Build 139)
http://protege.stanford.edu -->
```

Ontologie ParamsETSO

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="file:/C:/Protege_2.1/ParamsETSO.owl#"
  xml:base="file:/C:/Protege_2.1/ParamsETSO.owl">
  <owl:Ontology rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl"/>
  <owl:Class rdf:ID="Null">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >parametre non matche</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#others"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
  <owl:Class rdf:ID="FinalScheduleValidation">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ImbalanceReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="BalanceValidation">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AcknowledgementReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AnomalyReport">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="FinalSchedule">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="others">
    <rdfs:subClassOf>
      <owl:Class
rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#Parameters"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ValidationDone">
    <rdfs:subClassOf>
      <owl:Class rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="IO">
    <rdfs:subClassOf>
      <owl:Class
rdf:about="file:/C:/Protege_2.1/ParamsETSO.owl#Parameters"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ConvID">
    <rdfs:subClassOf
rdf:resource="file:/C:/Protege_2.1/ParamsETSO.owl#others"/>
  </owl:Class>
  <owl:Class rdf:ID="Parameters">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Parametres pour l'ontologie ETSO
</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="ScheduleMessage">
    <rdfs:subClassOf
rdf:resource="file:/C:/Protege_2.1/ParamsETSO.owl#others"/>
  </owl:Class>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.2 beta, Build 139)
http://protege.stanford.edu -->
```