Document :	*Spécification du langage logique de description contextuelle*
Sous-tâches :	3.1
Numéro des Livrables :	D3.1
Date	10/09/2010
Rédacteurs :	*Fabrice Jouanot (LIG), Anis Benyelloul (LIG), Marie-Christine Rousset(LIG)*
Coordinateurs :	Fabrice Jouanot

Sommaire

1 Introduction

1.1 Rappel des objectifs de la tâche 3

Cette tâche a pour objectif de prendre en compte l'hétérogénéité de la description du contexte ainsi que des composants et de leurs fonctionnalités pour le déploiement d'applications interactives dans lesquelles les dispositifs utilisables se découvrent, se connectent et interagissent à la volée et de façon décentralisée.
Dans ce projet, on distingue l'hétérogénéité de l'interopérabilité que l'on suppose acquise grâce à l'utilisation de formats et de protocoles permettant de faire communiquer et interagir des dispositifs physiques ou des composants logiciels.
Nous voulons permettre une description à haut niveau d'abstraction du contexte, des composants et de leurs fonctionnalités :

- **suffisamment compacte** pour que chaque dispositif puisse stocker sa description et télécharger les descriptions d'autres dispositifs, ainsi que des informations contextuelles,
- **suffisamment riche** pour qu'on puisse raisonner sur ces informations contextuelles et fonctionnelles afin par exemple d'inférer des propriétés d'alignement en vue de pouvoir qualifier différents assemblages de composants, en fonction de la demande de l'utilisateur et/ou du contexte.

La tâche 3 s'organise autour des quatre sous-tâches suivantes :

- T3.1 : la définition d'un langage logique de description contextuelle de composants et de services
- T3.2 : la découverte d'assemblages et d'alignements
- T3.3 : le raisonnement sur les propriétés et la continuité des services
- T3.4 : la réalisation d'un démonstrateur

Ce document s'intéresse particulièrement à la sous-tâche 3.1 et présente aussi les premiers éléments des sous-tâches 3.3 et 3.4.

1.2 Structure du document

Ce document se propose d'expliciter l'usage d'ontologies pour représenter et manipuler les informations contextuelles. L'idée directrice est de s'inspirer des acquis du Web Sémantique pour les adapter aux contraintes de la représentation du contexte en informatique ambiante. La première partie du document présente les différents moyens de représenter les ontologies dans le Web Sémantique. La seconde partie expose notre proposition de langage pour décrire un contexte sous la forme d'une ontologie et pour y exprimer des requêtes. Cette proposition est consolidée par la description du démonstrateur **Conquer** déjà livré.

2 Usage des ontologies pour composer avec des informations hétérogènes

Les ontologies jouent aujourd'hui un rôle déterminant pour le Web Sémantique grâce à la description et l'exploitation d'annotations sémantiques. La vision d'une architecture distribuée et interconnecté à l'échelle mondiale où données et services peuvent inter- opérer facilement possède de nombreux points communs avec la problématique de l'informatique ubiquitaire et mobile où des services et dispositifs doivent aussi inter-opérer pour la réalisation continue d'une tâche. L'annotation sémantique d'une ressource web consiste en la définition de connaissances sur cette ressource en relation avec les termes d'une ontologie prédéfinie. Cette définition peut tout aussi bien être réutilisée pour l'annotation d'une ressource disponible dans le contexte d'une tâche spécifique. La suite de cette section détaillera l'utilisation des ontologies dans leur environnement naturel, le Web Sémantique.

Une ontologie est une description formelle procurant à des utilisateurs humains un moyen de compréhension commun à propos d'un domaine spécifique, tout en restant interprétable et exploitable par une machine grâce à un support logique possédant naturellement des capacités à raisonner.
Les ontologies sont utiles à différentes fins, de l'organisation des données pour optimiser la navigation en passant par l'amélioration et la facilité de rechercher des informations sur le Web. De manière générale les ontologies sont utilisées comme liens sémantiques entre des sources d'informations hétérogènes. Elles sont la clé pour résoudre les problèmes d'hétérogénéités.
L'aspect le plus important des ontologies réside dans sa possibilité d'être au cœur d'un moteur d'inférence au sein des applications ou architectures. L'inférence la plus simple est déjà capable de produire des résultats dans le cadre de la gestion et de l'intégration de données : les réponses implicites à une requête peuvent devenir explicites. Ainsi les requêtes peuvent être représentées sous une forme générale ou plus spécifique afin de mieux s'adapter aux besoins. Les incohérences entre les sources de données peuvent aussi être découvertes et exploitées à des fins, par exemple, d'optimisation.
Les langages utilisés pour décrire les ontologies peuvent être vus comme des fragments décidables de la logique du premier ordre, résultant ainsi de la longue tradition en Intelligence Artificiel de concevoir des modèles pour représenter la connaissance couplés à des algorithmes d'inférence.

Dans cette partie nous présentons les bases formelles des langages standards du W3C ou recommandés pour décrire des métadonnées et des ontologies : RDF [RDF], RDFS [RDFS] et dans une moindre mesure OWL [OWL]. Les expressions RDF, RDFS ou OWL peuvent être exprimées dans différents formats (la plupart du temps XML), mais le point le plus important est qu'elles puissent être interprétées par la théorie de modèle standard de la logique du premier ordre, permettant ainsi d'inférer de nouvelles expressions par implication logique.
Le test de l'implication logique entre formules a été longuement étudié et est connu pour être indécidable dans le cas de la logique du premier ordre. Cependant le problème devient décidable pour plusieurs fragments de cette logique du premier ordre, fragments qui prennent alors un rôle central dans le domaine de la représentation des connaissances.

Les logiques de descriptions, par exemple, sont des fragments décidables qui permettent d'exprimer et de raisonner sur des axiomes logiques complexes, prenant en compte les prédicats unaires et binaires. Ces logiques couvrent un large spectre de langages logiques à base de classe pour lesquels un raisonnement par implication logique est décidable avec une complexité variable en fonction des opérateurs autorisés dans le langage. Ces logiques de description sont à la base du langage OWL recommandé par le W3C pour décrire des ontologies.
Dans la suite nous illustrerons sur un exemple ce qu'est une ontologie et comment raisonner à partir de celle-ci en exploitant son contenu. Ensuite nous détaillerons les langages RDF(s) et OWL, puis nous les relierons aux logiques de description afin de bien montrer l'impact du choix des constructeurs sur la décidabilité et l'implémentation de ces langages.

2.1 L'ontologie par l'exemple

Une ontologie est une description formelle d'un domaine d'intérêt à partir d'un ensemble fini de concepts qui représentent les classes des objets pertinents pour ce domaine.
Si on prend l'exemple des classes *Staff*, *Students*, *Departments* et *Courses*, elles représentent des concepts importants partagés ou qui fond sens pour tout utilisateur quelque peu familier avec l'organisation d'une université. La déclaration d'une relation de sous classe entre les classes *C* et *C'* exprime l'inclusion entre l'ensemble des instances de *C* et des instances de *C'*. Définir que la classe *Professor* est une sous-classe de la classe *Academic Staff* exprime donc une connaissance partagée au sein de l'université : tous les professeurs sont membres du personnel académique. Définir une relation de sous classe entre *Academic Staff* et *Staff* exprime alors que tous les membres du personnel académique, dont les professeurs, appartiennent au personnel de l'université.
Il est assez fréquent de choisir une représentation graphique de cet ensemble de sous-classes sous la forme d'une hiérarchie de classes ou **taxonomie**. La figure 2.1 illustre une taxonomie pour le domaine universitaire.

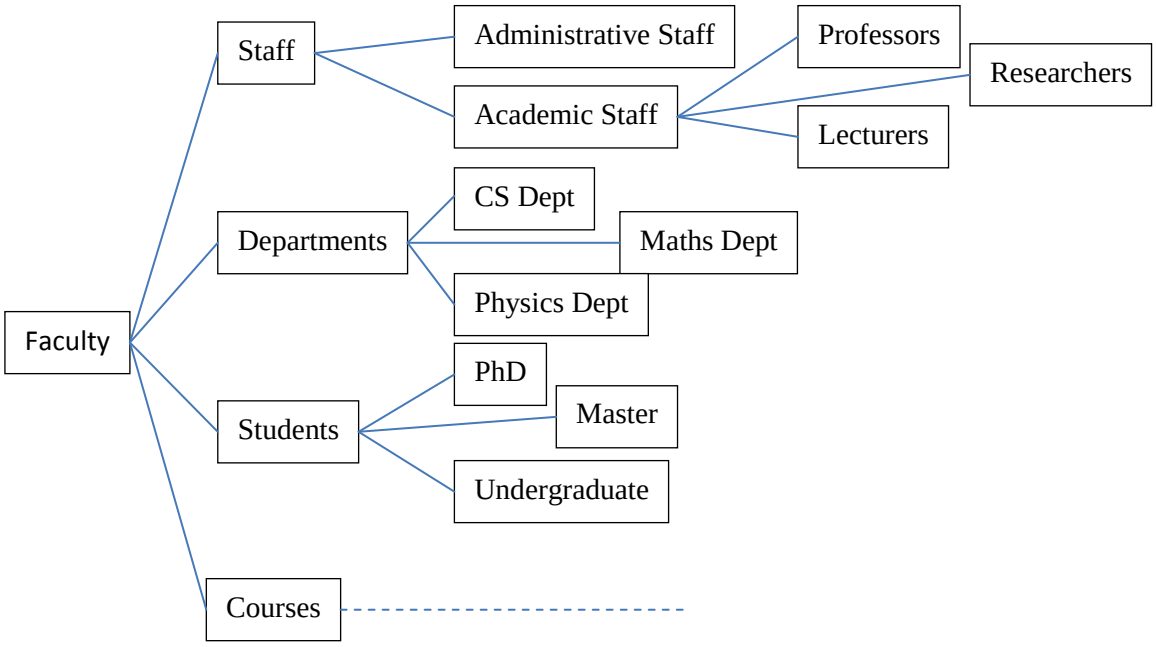

Figure 2.1 une hiérarchie de classes

Une ontologie peut inclure d'autres types de relation entre classes qui expriment alors certaines propriétés et connaissances qui traduisent des contraintes connues entre ces propriétés. Une

ontologie qui repose sur la hiérarchie de classes de la figure 2.1 pourrait introduire les relations suivantes :

- La déclaration de propriétés
  - *Teaches(Academic Staff, Courses)* signifie que si "*X* enseigne *Y*" est déclaré alors *X* appartient au personnel académique et *Y* est obligatoirement un cours.
  - *TeachesTo(Academic Staff, Students)* signifie que si "*X* enseigne à *Y*" est déclaré alors *X* appartient au personnel académique et *Y* est obligatoirement un étudiant.
  - *Manager(Staff, departments)* signifie que si "*X* est le directeur de *Y*" est déclaré alors *X* appartient au personnel et *Y* est un département.
- Des propriétés de disjonction entre classes, comme entre *Students* et *Staff*, puisqu'un étudiant ne peut pas appartenir au personnel.
- L'expression de contraintes de domaine qui expriment, par exemple, que seuls les professeurs et les maîtres de conférences peuvent intervenir auprès des étudiants de premier cycle, ou encore que chaque département doit posséder un directeur qui soit professeur.

Chaque relation entre classes et contraintes possèdent une définition logique formelle sur laquelle peut s'appuyer le raisonnement. Si *Dupond* est le directeur de *Infodept*, il est peut être inférer logiquement que *Dupond* est un professeur et que *InfoDept* est un département.
Des relations entre classes peuvent aussi être inférées. Prenons l'exemple où nous ajoutons à l'ontologie de la figure 2.1 une relation de sous-classe entre *PhD Students* et *Lecturers*, il devient possible d'inférer que *PhD Students* est aussi une sous-classe de *Staff*.
Cependant, en considérant la relation de sous-classe entre *PhD Students* et *Students*, et la relation de disjonction entre *PhD Students* et *Lecturers*, il est inféré que la classe *PhD Students* doit être vide ce qui dénote un problème au sein de l'ontologie. Il est bien sur possible de corriger cette anomalie en modifiant certaines déclarations définissant l'ontologie.
Il ne s'agit bien sûr que de quelques exemples issus d'un espace de données décrit, sémantiquement, à l'aide d'une ontologie. Le problème est de définir des algorithmes d'inférence qui soient correctes et complets vis-à-vis des paradigmes de la logique: ces algorithmes doivent inférer toutes les informations implicites (données ou connaissances) dérivés des faits avérés, des relations et des contraintes déclarées dans l'ontologie.

RDF (Ressource Framework Language) est un langage simple dédié à la description des métadonnées sur des ressources Web identifiées par des URIs. Une ontologie devient alors un ensemble d'expressions RDFS ou OWL qui ajoute des contraintes au vocabulaire RDF utilisé pour définir les métadonnées sur les classes et les propriétés. Les sections suivantes décrivent les langages RDF, RDFS et OWL en utilisant une syntaxe compacte plus adaptée que l'aspect verbeux de la notation XML classique.

2.2 RDF

En RDF, les métadonnées sur les ressources Web sont exprimées sous la forme d'un ensemble de triplets. Un triplé est composé d'un sujet, d'un prédicat et d'un objet. Il définit une relation portée par le prédicat entre le sujet et l'objet. Dans un triplé le sujet comme le prédicat est une URI pointant sur une ressource Web, alors que l'objet peut être soit une URI soit un simple littéral représentant

une valeur. Un triplé exprime donc qu'un sujet donné possède une valeur donnée pour une propriété donnée.

URIs (Uniform Ressource Identifier) et espace de nommage. Une URI est un identifiant global pour une ressource web: c'est simplement une URL qui permet à plusieurs utilisateurs (humains ou non) de référencer la même ressource. Une ressource prend alors un sens très large: une page web, un service web, un simple littéral dénotant le nom d'un individu, une chose, un concept ou une propriété. Une URI se décompose en deux parties, la première concerne l'espace de nommage et la seconde un identifiant local. La convention voulant que l'espace de nommage (URL) dispose d'une abréviation pour simplifier la lecture et la manipulation. Le W3C, entre autre, a défini un certain nombre d'espace de nommage standardisé dont les espaces rdf:, rdfs: et owl:. Cet aspect n'étant pas particulièrement important pour l'aspect gestion des connaissances, nous utiliserons dans la suite du document la notation simplifiée *:Name* pour référencer un individu, un concept ou un prédicat.

La syntaxe RDF. Elle permet avant tout de distinguer parmi toutes les ressources celles qui définissent des propriétés à l'aide des mots clés **rdf:type** et **rdf:Property**. Il est ainsi possible d'expliciter que *:Manager* est le nom d'une propriété à l'aide du triplé *:Manager rdf:type :Property*.
Un ensemble de triplés peut être représenté sous la forme d'un tableau ou d'un graphe. La représentation sous forme de graphe orienté est particulièrement pratique pour visualiser les informations reliées à un nœud individu. Dans ce type de graphe chaque triplé est représenté par un arc dirigé de son sujet vers son objet. L'exemple du tableau 2.1 et de la figure 2.2 illustre ces différentes représentations.

Sujet	Prédicat	Objet
:Dupond	:Manager	:InfoDept
:Dupond	:Teaches	:UE111
:Dupond	:TeachesTo	:Pierre
:Pierre	:EnrolledIn	:InfoDept
:Pierre	:RegisteredTo	:UE111
:UE111	:OfferedBy	:InfoDept

Tableau 2.1 un tableau de triplés RDF

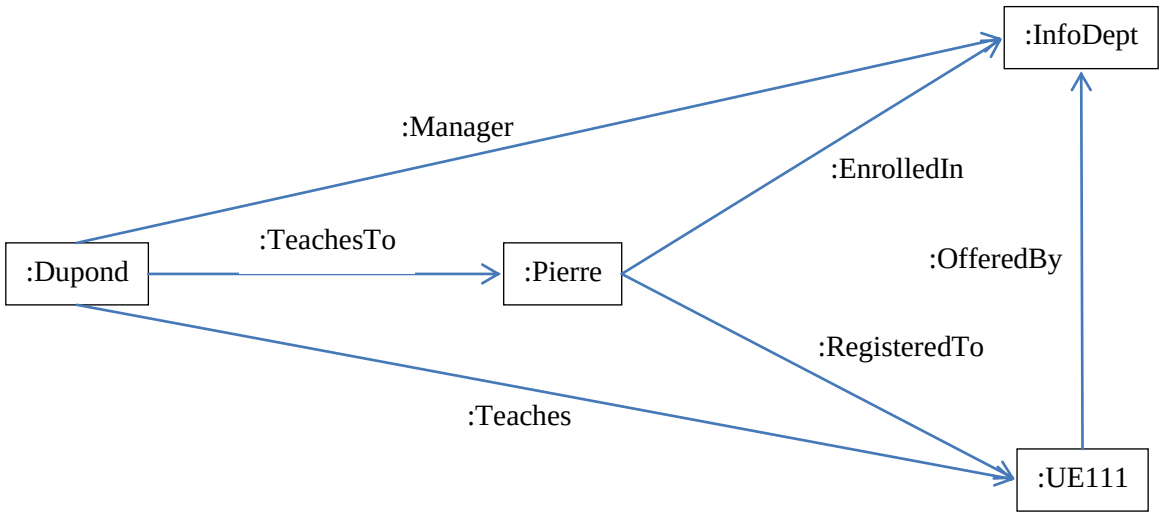

Figure 2.1 un graphe de triplés RDF

Noeuds anonymes. Un nœud anonyme est un sujet ou un objet dans un triplé RDF qui n'est pas identifié par une URI et qui ne correspond pas à un littéral. Il est noté _:p avec p un nom local pouvant être utilisé dans différents triplés pour déclarer différentes propriétés du nœud anonyme correspondant. Dans l'exemple suivant les triplés expriment le fait que :pierre connaît quelqu'un nommé "John Smith" ayant écrit un livre intitulé "Introduction to java":

:Pierre foaf:knows _:p1
_:p1 foaf:name "John Smith"
_:p1 :worte _:b1
_:b1 dc:title "Introduction to Java"

Les prédicats *foaf:knows* et *foaf:name* appartiennent au vocabulaire prédéfini FOAF, l'espace de nommage étant abrégé par l'expression *foaf:*. Ce vocabulaire permet de décrire des personnes avec leurs relations, leurs activités etc. Le prédicat *dc.title* appartient au vocabulaire *Dublin Core* pour décrire des documents.

La sémantique portée par RDF. Indépendamment des notions utilisées dans leur syntaxe, les triplés possèdent un formalisme logique issu de la logique du premier ordre et contenu dans la définition d'un triplé qui énonce qu'un sujet est mis en relation avec un objet par un prédicat. Un triplé s p o sans nœud anonyme s'interprète en logique du premier ordre comme un fait noté p(s,o), c'est-à-dire une formule atomique où p est le nom du prédicat, s et o jouent le rôle de constantes. Les nœuds anonymes sont interprétés comme des variables existentielles.
Ainsi un ensemble triplés RDF, avec ou sans nœuds anonymes, est une conjonction de littéraux dans leur forme positive et dans lesquels toutes les variables sont existentiellement quantifiées. L'exemple de Pierre ci-dessus peut s'écrire sous sa forme logique:
$\exists P1 \exists b1[knows(Pierre,p1) \wedge name(p1,"John\ Smith") \wedge wrote(p1,b1) \wedge title(b1,"Introduction\ to\ Java")]$

2.3 RDFS : RDF Schema

RDFS est langage pour définir des schémas, au-dessus de RDF. Il permet la déclaration de contraintes typées sur les objets, les sujets et les prédicats des triplés RDF. Les objets et les sujets sont déclarés en tant qu'instances de certaines classes. Les prédicats deviennent des propriétés.

En RDFS seuls les classes et propriétés atomiques sont permises. Les relations structurelles se limitent à l'inclusion entre deux classes ou propriétés atomiques, et à la restriction des domaines sources (sujet) et cibles (objet) d'un triplé.

La syntaxe RDFS. Les déclarations RDFS s'expriment sous la forme de triplés RDF utilisant des prédicats spécifiques comme mot clé et possédant une sémantique très précise. RDFS introduit, en plus de rdf:type, le prédicat rdfs:subClassOf pour exprimer les relations structurelles entre classes. Ce type d'information est une manière d'ajouter de la sémantique au vocabulaire RDF en plaçant des contraintes sur des termes dont l'interprétation peut être ambiguë (exemple :Java), contraintes qui associent ces termes à un terme plus général (exemple :CS Courses). Cette taxonomie de classes est exprimée par un ensemble de triplés de la forme C rdfs:subClassOF D.
Par exemple la hiérarchie de classe de la figure 2.1 peut se décrire en RDFS comme suit:

:Java rdfs:subClassOf :CS Courses
:AI rdfs:subClassOf :CS Courses
:BD rdfs:subClassOf :CS Courses
:CS Courses rdfs:subClassOf :Courses
:Logic rdfs:subClassOf :Courses
:Math Courses rdfs:subClassOf :Courses
:Algebra rdfs:subClassOf :Math Courses
:Probabilities rdfs:subClassOf :Math Courses

La sémantique portée par RFDS. Les expressions RDFS, construites sur RDF, peuvent s'interprétées par des formules de la logique du premier ordre. RDFS est en fait un fragment de *DL-LiteR*, logique de description de la famille *DL-Lite* [CGL 07]. Le tableau 2.2 donne l'équivalence entre les différentes notations.

RDFS	Logique premier ordre	DL
I rdf:type C	C(i)	I:C
C rdfs:subClassOf D	$\forall X(C(X) \Rightarrow D(X))$	$C \subseteq D$
P drfs:subPropertyOf R	$\forall X \forall Y(P(X,Y) \Rightarrow R(X,Y))$	$P \subseteq R$
P drfs:domain C	$\forall X \forall Y(P(X,Y) \Rightarrow C(X))$	$\exists P \subseteq C$
P rdfs:range D	$\forall X \forall Y(P(X,Y) \Rightarrow D(Y))$	$\exists P \subseteq D$

Tableau 2.2 : La sémantique RDFS

L'aspect opérationnel de la sémantique portée par ces constructeurs est réalisé par des règles d'inférence prévues pour certains modèles de triplés. Prenons l'exemple de la règle d'inférence qui supporte la déclaration rdfs:subClassOf en considérant les triplés de la forme:
R rdf:type A et A rdfs:subClassOf B alors le triplé r rdf:type B est inféré.
Cette règle peut être à son tour appliqué tant que des triplés correspondent aux modèles définis.
Une inférence similaire est définie pour appliquer la sémantique portée par le constructeur rdfs:subpropertyOf: Si r P s et P rdfs:subPropertyOf Q alors r Q s.
Des règles d'inférence sont aussi spécifiées pour les constructeurs rdfs:domain et rdfs:range qui posent des contraintes sur les propriétés:
Si P rdfs:domain C et x P y alors x rdf:type C
Si P rdfs:range D et x P y alors y rdf:type D
Des règles d'inférences sont ajoutées pour prendre en compte la combinaison des règles contraignant les propriétés avec celles définissant la taxonomie de classes:

Si P rdfs:domain A et A rdfs:subClassOf B alors P rdfs:domain B
Si P rdfs:range C et C rdfs:subClassOf D alors P rdfs:range D

En rendant opérationnel la sémantique d'un langage logique, le problème consiste à prouver sa complétude tout en respectant son implémentation. Prouver la complétude pour les règles énoncées ci-avant, en tenant compte de la logique du premier ordre sous-jacente, est trivial tant que ces règles restent des variantes syntaxiques des implications logiques décrites dans le tableau 2.2. La seule subtilité réside dans les deux dernières règles d'inférences :
$\forall X \forall Y(P(X,Y) \Rightarrow A(X)), \forall X(A(X) \Rightarrow B(X)) \vdash \forall X \forall Y(P(X,Y) \Rightarrow B(X))$
$\forall X \forall Y(P(X,Y) \Rightarrow C(Y)), \forall X(C(X) \Rightarrow D(X)) \vdash \forall X \forall Y(P(X,Y) \Rightarrow D(Y))$

Différents outils existent aujourd'hui pour stocker, gérer et interroger des données RDFS, le plus connu et le plus flexible restant Jena [Jen]. Ces outils utilisent généralement les expressions RDFS pour saturer la base de triplés avec toutes les déclarations possibles inférables. Ainsi il devient possible d'interroger la base de triplés avec des langages de haut niveau comme SPARQL [W3C08]. Nous étudierons dans la section 3 les spécificités à considérer dans le domaine de l'informatique ambiante et dans la représentation du contexte en utilisant ces techniques de représentation de la connaissance.

2.4 OWL

En OWL des classes complexes peuvent être construites à partir de classes et propriétés atomiques en utilisant des constructeurs empruntés à la logique de description. De plus l'expression d'inclusion généralisée permet la déclaration de contraintes plus expressives qu'en RDFS.
De manière très générale OWL étend RDFS avec l'expression de contraintes impossibles en RDFS. Les déclarations OWL sont encore exprimés à l'aide de triplés RDF, mais utilisent des prédicats spécifiques qui supportent une sémantique plus riche.
OWL fournit des constructions très flexibles et expressives pour modéliser les informations au sein du Web Sémantique. La combinaison de ces constructeurs entre eux et avec ceux de RDFS permet la définition de contraintes qui expriment des relations complexes entre les propriétés, classes et individus. Cependant disposer d'une sémantique opérationnelle, sous la forme de règles d'inférences, pour tous les constructeurs et en tenant compte de toutes les interactions possibles entre ces constructeurs reste un problème ouvert.
Mais comme la majorité des constructeurs OWL sont issus de la logique de description, et par la même, possèdent une description en logique du premier ordre, les résultats sur le raisonnement obtenu en logique de description sont réutilisables. Il devient possible de comprendre si les inférences sur des triplés RDF en considérant des contraintes OWL sont réalisables, en particulier sur le plan de la décidabilité et de la complexité des algorithmes. Ces résultats sont particulièrement importants pour étudier l'impact de nouveaux constructeurs dans notre langage de description du contexte.
Dans la suite de cette section nous passons en revue les différents constructeurs OWL et leurs équivalents dans les différentes notations.

L'expression de la disjonction entre classes. OWL dispose du prédicat owl:disjointWith pour exprimer que deux classes C et D sont disjointes. La contrainte de disjonction se trouve être assez

naturel dans nombre d'applications. En reprenant l'exemple universitaire, les classes Students et Staff peuvent être déclarées disjointes avec le triplé :Students owl:disjointWith :Staff. Le tableau ci-après donne l'équivalence des notations; à noter l'apparition de la négation qui a un impact direct sur l'aspect opérationnel de ce constructeur.

OWL	Logique premier ordre	DL
C owl:disjointWith D	$\forall X(C(X)\Rightarrow\neg D(X))$	$C\subseteq\neg D$

Constructeurs pour les propriétés. OWL autorise la déclaration d'une propriété inverse d'une autre propriété. La sémantique logique est donnée dans le tableau ci-dessous:

OWL	Logique premier ordre	DL
P owl:inverseOf Q	$\forall X\forall Y(P(X,Y)\Leftrightarrow Q(Y,X))$	$P\equiv Q$
P rdf:type owl:SymmetricProperty	$\forall X\forall Y(P(X,Y)\Rightarrow P(Y,X))$	$P\subseteq P$
P rdf:type owl:FunctionalProperty	$\forall X\forall Y(P(X,Y)\wedge P(X,Z))\Rightarrow Y=Z$	$(\text{func}P)\text{or}\exists P\subseteq(\leq 1P)$
P rdf:type owl:InverseFunctionalProperty	$\forall X\forall Y(P(X,Y)\wedge P(Z,Y))\Rightarrow X=Z$	$(\text{func}P^{-})\text{or}\exists P^{-}\subseteq(\leq 1P^{-})$

Une propriété donnée peut être fonctionnelle ou symétrique. Toujours dans l'exemple universitaire, la contrainte qui énonce qu'un département ne peut avoir qu'un et un seul directeur s'exprime sous la forme du triplé :Manager rdf:type owl:InverseFunctionalProperty qui s'ajoute aux triplés existants.

L'expression de l'équivalence entre classes et entre propriétés. Les constructeurs owl:equivalentClass et owl:equivalent Property définissent l'équivalence entre deux classes ou deux propriétés. Formellement ces deux constructeurs n'apportent pas plus d'expressivité au langage, la même sémantique peut s'exprimer en RDFS avec deux triplés:
C owl:equivalentClass D est équivalent aux triplés C rdfs:subClassOf D et D rdfs:subClassOf C.

L'expression de restriction sur les propriétés. La caractéristique principale d'OWL est de pouvoir créer de nouvelles classes à partir de déclarations de restrictions sur des classes et propriétés existantes. Il devient ainsi possible de définir des contraintes complexes du genre "chaque département possède un directeur qui doit être professeur", "seuls les professeurs et les maitres de conférences peuvent enseigner aux étudiants de premier cycle".
Une expression owl:Restriction représente une classe spéciale associée à un nœud anonyme, et à des restrictions sur les propriétés (someValuesFrom, allValuesFrom,minCardinality, maxCardinality) décrivant plus précisément la classe.
Dans l'exemple ci-dessous les triplés déclarent l'ensemble des individus dont toutes les valeurs de la propriété directeur sont issues de la classe professeur:
_:a rdfs:subClassOf owl:Restriction
_:a owl:onProperty :Manager
_:a owl:allValuesFrom :Professors
Il devient alors trivial de formaliser la contrainte " chaque département possède un directeur qui doit être professeur": :Departements rdfs:subClassOf _:a
Les déclarations owl:minCardinality and owl:maxCardinality imposent des contraintes sur le nombre d'individus pouvant être associé par une propriété P. Prenons l'exemple de l'ensemble suivant de triplés qui décrivent la classe d'individus ayant au moins 3 inscriptions et celle ayant au plus 6 inscriptions:
_:a rdfs:subClassOf owl:Restriction

_:a owl:onProperty :RestereredTo
_:a owl:minCardinality 3
_:b rdfs:subClassOf owl:Restriction
_:b owl:onProperty :RestereredTo
_:b owl:maxCardinality 6
La déclaration de la contrainte "les étudiants doivent s'inscrire à au moins 3 cours et au plus 6" s'exprime alors simplement:
:Students rdfs:subClassOf _:a
:Students rdfs:subClassOf _:b

Il est bien sur possible d'exprimer ces constructeurs de restriction en logique du premier ordre et en logique de description:

OWL	Logique premier ordre	DL
a owl:onProperty P a owl:allValuesFrom C	$\forall Y(P(X,Y)\Rightarrow C(Y))$	$\forall P,C$
a owl:onProperty P a owl:oneValuesFrom C	$\exists Y(P(X,Y)\wedge C(Y))$	$\exists P,C$
a owl:onProperty P a owl:minCardinality n	$\exists Y_1 \ldots \exists Y_n(P(X,Y_1)\wedge \ldots \wedge P(X,Y_n)\wedge(\bigwedge_{i,j\in[1..n]i\neq j}(Y_i\neq Y_j))$	$(\geq nP)$
a owl:maxCardinality n	$(\exists Y_1 \ldots \exists Y_n \exists Y_{n-1}\ P(X,Y_1)\wedge \ldots \wedge P(X,Y_n)\Rightarrow \bigvee_{i,j\in[1..n]i\neq j}(Y_i\neq Y_j)$	$(\leq nP)$

Union et intersection. Le constructeur owl:intersectionOf permet de définir l'appartenance d'instances à deux classes. owl:unionOf exprime que les instances appartiennent à n'importe laquelle de deux classes. Par exemple la contrainte "seuls les professeurs et les maitres de conférences peuvent enseigner aux étudiants de premier cycle" requière le constructeur d'union:
_:a rdfs:subClassOf owl:Restriction
_:a owl:onProperty :TeachesTo
_:a owl:oneValuesFrom :Undergraduate Students
_b: owl:unionOf (Professors Lecturers)
_:a rdfs:subClassOf _b
La sémantique de ces constructeurs s'exprime comme suit en logique du premier ordre et logique de description:

OWL	Logique premier ordre	DL
owl:intersectionOf(C D …)	$C(X)\wedge D(X)\ldots$	$C\cap D\ldots$
owl:unionOf(C D …)	$C(X)\vee D(X)\ldots$	$C\cup D\ldots$

Il existe actuellement très peu d'outils qui supportent le raisonnement sur les déclarations OWL. Les seuls outils disponibles sont de type logique de description: Fact [FAC], RACER [RAC], Pellet [SPG 07].

2.5 Les bases de la logique de description

Il apparaît donc que la logique du premier ordre constitue la fondation du langage owl. La logique du premier ordre (ou logique de prédicats) est tout particulièrement appropriée pour la représentation des connaissances et raisonner sur ces connaissances. D'un point de vue logique, les classes sont des

prédicats unaires, les propriétés sont des prédicats binaires et les contraintes sont des formules logiques définies sous forme d'axiomes sur ces prédicats.
Le problème du raisonnement automatique en logique du premier ordre a été très étudié. Il en ressort que cette logique est semi-décidable. Il n'existe donc aucun algorithme de raisonnement type qui, appliqué à deux formules ϕ et ψ, soit capable de vérifier si ϕ est l'implication de ψ ou non. Même le problème, apparemment, plus simple consistant à vérifier si une formule logique est satisfiable est semi-décidable. En fait les deux problèmes peuvent être réduits l'un à l'autre, et la logique du premier ordre reste semi-décidable.
Les recherches se sont concentrées sur la construction de fragments décidables de la logique du premier ordre, c'est-à-dire la définition d'un sous-ensemble de formules pour lesquelles l'implication entre formules est vérifiable automatiquement par un algorithme. Les logiques de description (DLs) sont justement des fragments décidables permettant de raisonner sur des axiomes logiques complexes portant sur des prédicats unaires et binaires. C'est exactement la sémantique opérationnelle requise pour gérer des ontologies, et cela justifie l'emprunt aux DLs des constructeurs du langage owl.

Les logiques de description composent un large choix de langage logique à base de classes, dont la complexité des raisonneurs varie en fonction des constructeurs autorisés par le langage. Une base de connaissance en DL est composée d'une partie intentionnelle (Tbox) et un ensemble de faits (Abox). Les problèmes de raisonnement qui sont bien étudiés sont la vérification de cohérence de bases de connaissances, la cohérence des instances et la vérification de la propriété de subsumption. Ces deux dernières vérifications se ramènent à la vérifier l'incohérence pour les DLs disposant de la négation. Les travaux sur la DLs reposent toujours sur un compromis entre le pouvoir d'expression du langage et la complexité algorithmique d'un raisonnement correct et complet [BCM 03]. Rattaché le langage de modélisation d'ontologie owl aux logiques de description est de première importance lorsqu'il s'agit de calculer formellement le prix à payer pour gérer automatiquement des contraintes complexes. Le fragment owl qui correspond aux logiques de description est souvent nommé OWL DL. Il s'agit en fait d'un variante syntaxique de SHOIN DL [HPSvH03] qui est obtenu en ajoutant à ALC DL (logique de description de minimale) la notion de cardinalité ($\geq$nP), d'unicité {a}, et de rôle inverse P^- (rôles atomiques).
En pratique, les raisonneurs existants comme FaCT, RACER et Pellet possèdent des performances raisonnables tant que les ontologies conservent une taille raisonnable. Les ontologies utilisées dans des applications réelles ne sont que très rarement concernées par des contraintes complexes. Cependant la représentation du contexte de manière similaire aux ontologies, est d'avantage concernée par les problèmes de passage à l'échelle et de complexité des contraintes.

3 Représentation du contexte sous la forme d'une ontologie

L'équipe Hadas du LIG propose un modèle du contexte qui s'appuie sur les standards du web sémantique. L'approche retenue représente un contexte d'exécution sous la forme d'une ontologie décrite dans le langage RDFS [RDFS]. L'idée principale est de pouvoir disposer d'une approche déclarative pour modéliser et interroger les données du contexte. La manipulation comme l'interrogation du contexte peuvent ainsi exploiter les propriétés de raisonnement et d'inférences utilisées dans le contexte du web sémantique.

3.1 Le modèle CONQUER

Notre modèle du contexte reprend donc la définition d'une ontologie qui s'énonce comme la description formelle d'un domaine d'intérêt basé sur un ensemble fini de concepts représentant les classes des objets (instances) pertinents dans le domaine. Il est constitué de quatre classes de base dans le domaine de l'informatique ambiante, relatives aux concepts de Personne, Dispositif, Service et Tâche [Benyelloul10]. Chacune des classes est composée d'un ensemble de propriétés inspiré du modèle 5W1H [Jang05]. Ce noyau, constituant une métareprésentation du contexte (**Figure 1**), peut être facilement enrichi par spécialisation des classes de base afin de capturer toute la sémantique nécessaire au contexte d'une application.

- La classe **Person** regroupe les utilisateurs participant à un scénario applicatif. Ces utilisateurs interagissent avec des dispositifs ou initient des tâches. Ils peuvent se déplacer dans l'environnement et possèdent des postures ou gestuelles pertinentes. Trois types de propriétés permettent de capturer ce comportement: **What** caractérise le dispositif avec lequel une personne interagit, **Where** spécifie sa localisation courante, **How** modélise le comportement physique de la personne.
- La classe **Device** représente les dispositifs physiques disponibles dans l'environnement qui sont décrits par les services qu'ils offrent (propriété **offers**) et leur localisation (**where**).
- La classe **Service** représente les services logiciels offerts par des dispositifs ou requis pour la réalisation d'une tâche. Un service ne possède pas de propriétés au niveau méta.
- La classe **Task** représente les objectifs à atteindre et requiert un ensemble de dispositifs nécessaires à la réalisation de chaque tâche (propriété **What**).

La **Figure 1** donne la représentation graphique du métamodèle Conquer du contexte et repose sur RDFS. Une ontologie, et donc la description d'un contexte sous Conquer, est un ensemble de règles en RDFS, c'est-à-dire un ensemble de contraintes sur un vocabulaire exprimé en RDF et permettant d'exprimer la notion de classe, de propriétés, et les relations entre eux.

:Bob :What :BobSmarthphone
:Bob :Where :BobCar

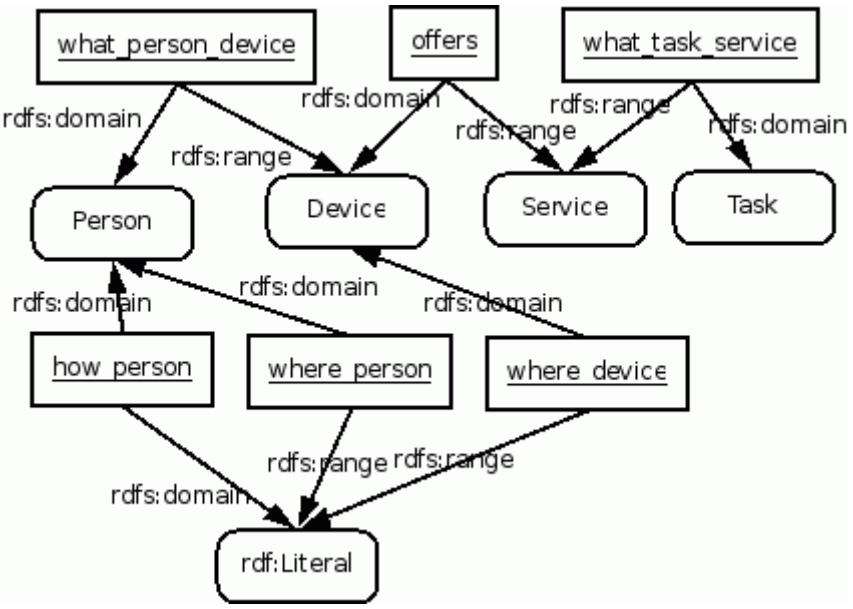

Figure 1. Métareprésentation du modèle de contexte en RDFS.

RDF fournit les bases pour la représentation de connaissances et la navigation dans un graphe de triplets. Cependant le modèle n'est pas assez contraint pour capturer la sémantique contenue dans une ontologie ou une représentation du contexte. RDFS apporte des contraintes pour décrire un schéma au-dessus des triplets RDF. La syntaxe RDFS reprend le modèle en triplet de RDF, en ajoutant de la sémantique via certains prédicats. Le prédicat **rdfs:type** permet de typer un objet en le déclarant comme instance d'une classe (:Bob rdf:type :person). Le prédicat **rdfs:subclassOf** permet de spécifier des relations entre les classes afin de construire une hiérarchie (:Smartphone rdf:subclassOf :Device), compléter de **rdfs:subpropertyOf** qui définit des relations entre propriétés. Une propriété se retrouve typée à l'aide des prédicats **rdfs:domain** et **rdfs:range** qui permettent de restreindre le domaine respectivement du sujet et de l'objet d'une propriété (:offers rdfs:domain :Device, :offers rdfs:range :Service). Plus encore, RDFS peut s'exprimer sous la forme d'axiomes d'un fragment de la logique de description [Baader07] DL-Lite [Calvanese07], propriété qui assure la terminaison des requêtes et la complétude des résultats.

La **Figure 2** illustre un modèle du contexte plus riche utilisant le noyau Conquer, ainsi que les instances définies au sein de ce contexte d'exécution. Cet exemple cible un scénario d'écoute de messagerie téléphonique d'un utilisateur : La personne Bob est localisée dans son véhicule avec deux dispositifs dans son environnement (aSmarPhone et aDashBoard) qui possèdent des capacités audio, une tâche d'écoute de message est en cours. Les relations sémantiques entre les différentes instances en présence vont évoluer en fonction des événements capturés ou de changement de situation : Bob en interaction avec son smartphone, avec le tableau de bord de son véhicule, etc. L'apparition et la disparition de relations entre les instances du monde physique et logiciel se traduiront par l'instanciation ou la suppression de contraintes RDFS.

L'exploitation des données du contexte nécessite cependant des outils pour mettre à jour ces données, interroger ces données et de manière plus générale pouvoir raisonner sur ces données. Nous proposons donc une méthode pour interroger les informations du contexte d'une manière efficace, capable de passer à l'échelle de l'informatique ambiante, et qui repose là encore sur les standards du Web sémantique.

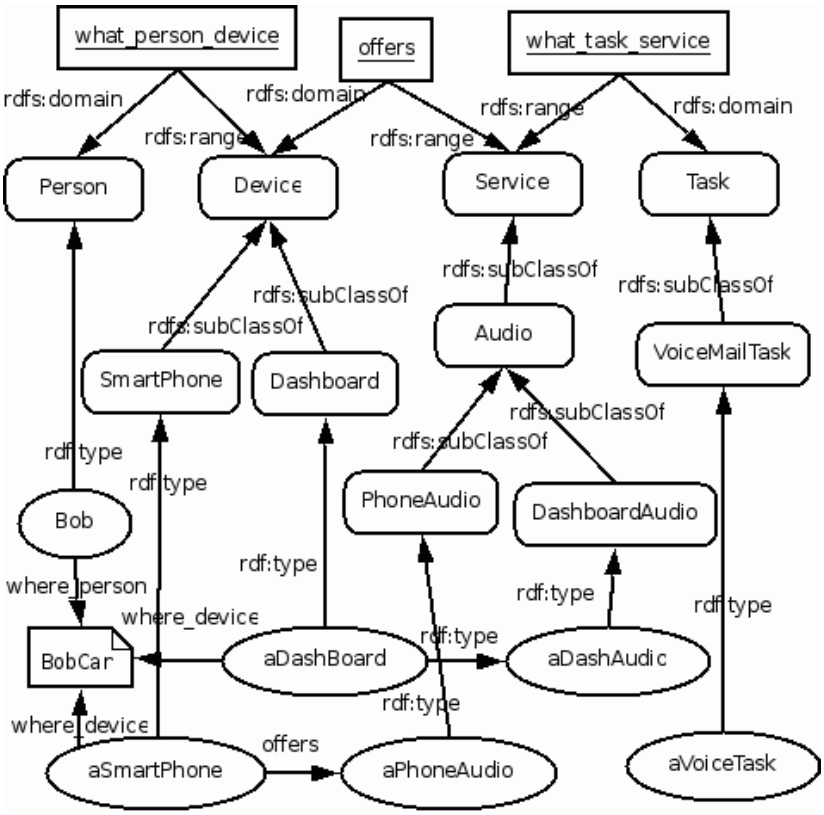

Figure 2. Exemple de modélisation d'un contexte.

3.2 Interroger le contexte

Notre modèle du contexte reposant sur RDFS et donc sur des données exprimées RDF, il est possible d'exprimer un contexte sous la forme d'un ensemble de faits sur des prédicats unaires et binaires. Si le contexte peut s'exprimer par les paradigmes de la logique du premier ordre, il est alors possible d'inférer de nouveaux faits et donc de poser des requêtes en suivant toujours les paradigmes de la logique.

Conquer permet d'écrire les requêtes exprimables sous la forme d'une conjonction de prédicats. Par exemple, la requête "quels sont les dispositifs offrant un service de type audio dans la voiture de Bob ?" peut être définie comme suit en logique du premier ordre:

$$q(x): \text{Device}(x) \wedge \text{where\_device}(x,\text{BobCar}) \wedge \text{offer}(x,s) \wedge \text{Audio}(s)$$

L'évaluation de cette requête sur les données du contexte consiste à trouver les instances pour les différentes variables pour lesquelles l'expression en logique du premier ordre est substituée par les faits connus. Sur l'ensemble des faits exprimés dans la **Figure 2**, l'expression logique ne peut être satisfaite et la requête n'a pas de réponse. Cependant, si on considère l'évaluation de la requête sur l'ontologie définie par le contexte, les axiomes d'inclusion suivants doivent être considérés :

$$\text{SmarPhone} \subseteq \text{Device} \quad \text{DashBoard} \subseteq \text{Device}$$
$$\text{PhoneAudio} \subseteq \text{Audio} \quad \text{DashBoardAudio} \subseteq \text{Audio}$$

De nouveaux faits sont alors inférés et l'évaluation de l'expression logique précédente donne le résultat {aSmarthPhone,aDashBoard}.

Conquer permet l'utilisation du langage d'interrogation déclaratif SPARQL [SPARQL] pour interroger les données du contexte. Ce langage est un standard du W3C pour interroger les données RDF (et donc RFDS). L'évaluation correcte des requêtes SPARQL est assurée par l'exécution du moteur SPARQL sur un ensemble de faits saturés. La requête logique précédente devient alors,

```
SELECT ?d WHERE { ?d type Device .
?d where_device BobCar .
?d offers ?s .
?s type Audio . }
```

Cette approche permet de poser, sous forme déclarative, des requêtes complexes sur les données du contexte. Les réponses à ces requêtes sont à la base de réactions adaptées au contexte.

Les déclarations RDFS forment un ensemble de règles qui peuvent être appliquées en utilisant un algorithme de chaînage avant. Cependant, l'évaluation de requêtes peut aussi être réalisée à l'aide d'un langage de requêtes standard (de type SQL par exemple) sur un ensemble de faits, à la manière d'une interrogation sur une base de données relationnelle. L'ensemble des faits doit englober à la fois les faits connus et les faits inférés (à partir des axiomes d'inclusion, entre autres, provenant de la définition du contexte). Cet ensemble de faits doit être maintenu en appliquant un algorithme de chaînage avant : soit de manière incrémentale, soit de manière globale et on parle alors d'algorithme de saturation.

3.3 Perspectives et extension du modèle

La sémantique du langage Conquer est directement issue de RDFS mais n'inclue aucun constructeur OWL permettant la définition de contraintes complexes. Le modèle est volontairement épuré pour assurer une gestion des requêtes optimales tout en supportant un passage à l'échelle des données du contexte. A la différence d'une ontologie classique, les données contextuelles sont nombreuses, changent très fréquemment, et apparaissent ou disparaissent aussi très rapidement. Il s'agit donc de pouvoir inférer sur une base de connaissance saturée, et donc de disposer d'un algorithme et d'une implémentation particulièrement efficace. La simplicité du modèle Conquer permet de répondre, dans un premier temps, à cette forte contrainte.

Néanmoins nous étudions la possibilité d'ajouter de nouveaux constructeurs, issus de OWL, qui se révèlent très utiles dans le cadre de la description de dispositifs et leur mise en contexte. Il s'agit de faire une analyse très fine sur l'impact de ces constructeurs dans le modèle actuel, et donc dans la performance de la saturation. Le livrable D3.2 est particulièrement important dans cette étape puisqu'il met en situation notre modèle à travers différents scénarios afin 1) d'illustrer l'aspect déclaratif de notre approche, 2) d'identifier les lacunes dans la modélisation.

Le modèle Conquer avec son approche déclarative tant sur l'aspect modélisation qu'interrogation est au cœur du processus de maitrise de l'hétérogénéité. Cet aspect sera abordé dans le livrable D3.3 qui mettra tout particulièrement en lumière la résolution de descriptions hétérogènes de dispositifs en exploitant la sémantique contenu dans la base de connaissance utilisée pour représenter le contexte.

4 Conquer

Nous avons développé un web service : Conquer, qui implémente une base de faits RDFS et offre les opérations suivantes pour manipuler ces faits :

- *Add* : Ajout d'un triplet dans la base ;
- *Delete* : Suppression d'un triplet ;
- *Query* : Exécution d'une requête SPARQL sur la base.

Le service maintient une base de faits RDFS « saturée » ce qui permet de réutiliser des moteurs de requête SPARL existants, qui n'implémentent pas, pour la plupart, d'algorithmes de chaînage avant.

Nous avons aussi développé une interface web dans le but d'illustrer les fonctionnalités du service, en fournissant une représentation graphique des données, et des formulaires simplifiant les opérations d'édition et d'interrogation. La **Figure 3** est une capture d'écran de la page d'accueil, montrant une représentation sous forme d'un graphe de la base de connaissance RDFS. La **Figure 4** montre un exemple d'utilisation d'un formulaire pour l'expression de requêtes, ainsi que l'affichage des résultats sous forme de sous-graphes.

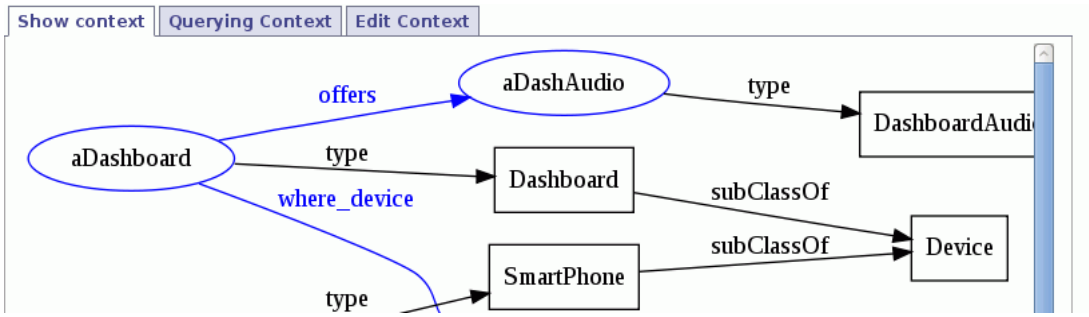

Figure 3. Représentation Graphique de Données du Contexte.

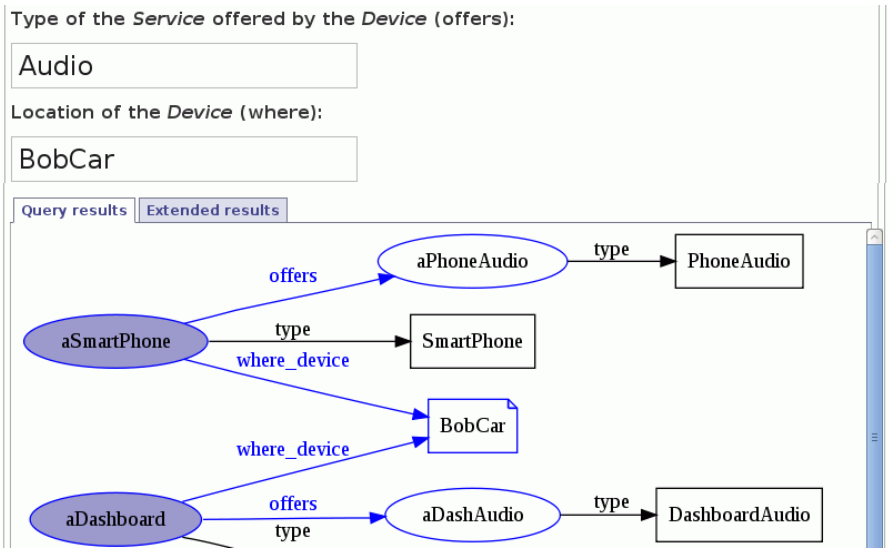

Figure 4. Exemple de Résultats de Requête.

De nombreux outils pour manipuler les ontologies existent et peuvent donc aussi représenter et gérer des informations du contexte exprimées en logique de description. Protégé [Protégé] permet

d'éditer des ontologies en OWL [OWL] et dispose de quantité de plug-ins pour exploiter les informations (raisonneur, solveur, prouveur). C'est l'outil idéal pour définir visuellement une ontologie complexe, constituée d'un grand nombre de classes. Mais dès que l'ontologie est sujette à de fréquentes modifications, ce genre d'outil dédié à la spécification devient peu utilisable. Le même problème apparaît avec les différents raisonneurs comme Racer [Racerpro], Pellet [Sirin07] ou Fact [Fact] dont les moteurs d'inférences réclament une ontologie stable.

L'informatique ambiante implique des environnements très dynamiques et donc une nécessité de remodeler sans cesse le contexte. La représentation, l'exploitation et l'interrogation de connaissances demande des outils souples et réactifs. A la différence de canevas logiciel très complet comme Jena [Jena] ou Quonto [Acciarri05], tous deux capables de raisonner sur des masses de données, Conquer est un outil plus simple mais bien adapté aux données très dynamiques du contexte, aux dispositifs légers, et facilement extensible en fonction de l'application ciblée.

5 Bibliographie

[BCM 03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider, editors. The description logic handbook: Theory, implementation, and Applications. Cambridge University Press, 2003.

[Benyelloul10] A. Benyelloul, F. Jouanot, M.C. Rousset. Conquer: an RDFS-based model for context querying. UbiMob'10, 6 èmes Journées Francophones Mobilité et Ubiquité - 7 au 9 juin 2010, Lyon, France.

[CGL 07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-lite family. Journal of Automated Reasoning, 39(3):385-429, 2007.

[FAC] Fact++, http://owl.cs.manchester.ac.uk/fact++.

[HPSvH03] I. Horrocks, P.F. Patel-Scheider, F. van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. Journal of Web Semantics, 4(2):144-153,2003.

[Jen] Jena – a semantic web framework for java, http://jena.sourceforge.net.

[OWL] OWL, http://www.w3.org/TR/owl-features/.

[RAC] Racerpro, http://www.racer-systems.com.

[RDF] RDF, http://www.w3.org/RDF.

[RDFS] RDFS, http://www.w3.org/TR/2004/REC-rdf-schema-20040210.

[SPG 07] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, Y. Katz. Pellet: A practical OWL-DL reasoned, Journal of Web Semantics, 5(2):51-53, 2007.