Ontologie et modèle relationnel pour le référencement, la recherche et l'agrégation des composants logiciels.

Anis Masmoudi
LICEF
100, rue Sherbrooke Ouest,
Montréal, Québec, H2X 3P2, Canada
1 514 843 2015 # 810939
anis.masmoudi@licef.ca

Roger Champagne
ÉTS
1100, rue Notre-Dame Ouest,
Montréal, Québec, H3C 1K3, Canada
1 514 396 8800 #
roger.champagne@etsmtl.ca

Gilbert Paquette
LICEF
100, rue Sherbrooke Ouest,
Montréal, Québec, H2X 3P2, Canada
1 514 843 2015 # 2292
gilbert.paquette@licef.ca

Résumé
Nos travaux visent à développer les systèmes sous la forme de composants en les agrégeant à l'aide d'un modèle décrivant leurs principales caractéristiques. Nous avons récemment proposé une structure de métadonnées des composants logiciels intitulée SOCOM (SOftware COmponent Metadata). Nous l'utilisons pour construire une banque de composants qui servira de support pour les futurs développements logiciels. L'article présente un modèle ontologique de SOCOM. Ensuite, nous avons choisi une implémentation de l'ontologie de SOCOM dans une architecture distribuée que nous comparons avec un autre modèle relationnel de SOCOM. Cette implémentation nous aide à référencer et à rechercher des composants dans le système TELOS[1] en cours de construction au sein du réseau LORNET (Learning Object Repositories' NETworks). À la fin, nous décrivons l'interrogation de l'ontologie SOCOM pour la recherche de composants mettant en œuvre des spécifications DIG et des moteurs d'inférences qui s'appuient sur la logique descriptive.

Catégories et les descripteurs des sujets
D.2.13 [**Reusable Software**]: Domain engineering, Reusable libraries, Reuse models.

Termes Généraux
Documentation, Performance, Design, Reliability, Experimentation, Theory.

Keywords
Composant logiciel, Ontologie, Développement basé sur les composants, Modélisation, Métadonnées de composants logiciels, moteur d'inférence, Spécifications DIG.

1. INTRODUCTION
La construction des systèmes informatiques et les développements logiciels se basent de plus en plus sur le composant comme unité de base. Ces composants doivent être référencés pour qu'ils puissent être réutilisés. Ce référencement doit couvrir plus d'un aspect afin d'obtenir une description globale du composant et pouvoir décider des différents scénarios de son agrégation avec d'autres composants. Plusieurs plateformes industrielles référencent les composants en tenant compte essentiellement de leurs caractéristiques techniques. Cependant, d'autres caractéristiques non fonctionnelles et métiers du composant peuvent informer des possibilités de sa réutilisation. Ainsi, nous retrouvons dans ces concepts le domaine du développement à base de composants [1]. C'est dans ce même contexte que s'inscrit le projet LORNET[2]. Un de ses objectifs principaux est la mise en place d'un système générateur de systèmes d'apprentissage et, de façon plus générale, de systèmes de gestion des connaissances. Ces systèmes à leur tour génèreront des applications spécifiques conformément à une discipline scientifique ou à un domaine de travail. Ces applications sont accessibles par les tuteurs (chargés de cours) pour assister les apprenants lors du suivi d'un cours à distance ou lors de la participation à un processus de travail. Chaque apprenant utilise des ressources pédagogiques ou de support au travail et en fournit d'autres. Cette cascade, supportée par le système TELOS (Tele-Learning Operating System) du projet LORNET, est conçue par agrégation de composants logiciels et de ressources existantes.

Dans cet article, nous rappelons la structure des métadonnées des composants logiciels SOCOM (SOftware COmponent Metadata) que nous avons présentée ailleurs dans [7]. Nous présentons brièvement la nouvelle structure étendue de SOCOM (Section 3.). Puis nous expliquons la modélisation graphique de l'ontologie de SOCOM à la section 4. La section 5. décrit les outils et les langages nécessaires pour la modélisation et l'implémentation de ce modèle ontologique selon le standard OWL (Ontology Web Language). Ensuite, nous introduisons au début de la section 6. les logiques descriptives et les spécifications DIG (Description Logics Implementation Group) qui permettent la représentation en XML de cette ontologie. Dans la sous-section 7. , nous décrivons notre implémentation de l'ontologie SOCOM dans une architecture distribuée qui s'appuie sur les paradigmes précédemment définis. Puis, nous justifions, dans la sous section 8. , notre choix de l'implémentation de SOCOM par une comparaison, à l'aide d'un exemple, avec une deuxième implémentation relationnelle. À la fin, nous rappelons les idées maîtresses de cet article et nos perspectives de développement.

2. TRAVAUX CONNEXE
Le développement logiciel à base de composants (CBSD[3]) consiste à construire des systèmes et des logiciels complexes en

[1] Voir [14],[6]

[2] LORNET: www.lornet.org

[3] CBSD : Component-Based Software Development

intégrant des unités existantes, soit des composants. Cette pratique vise à améliorer la flexibilité et la maintenabilité de ces systèmes. Elle aide aussi à réduire les coûts du développement et la mise à jour des systèmes de grande taille. Par définition, un composant logiciel est une unité de composition reliée à des interfaces spécifiques et se rattachant seulement à un contexte de dépendances explicite. Un composant logiciel peut être déployé indépendamment et faire l'objet d'une composition ou agrégation avec d'autres composants par un tiers [20].

Contrairement au développement classique, l'agrégation des composants devient la pièce maitresse de l'approche CBSD. Par conséquent, l'agrégation est une approche à prendre en considération au moment de la prise de décision quant à l'acquisition, la réutilisation ou la construction de composant. Pour ce faire, il est nécessaire de mener les activités suivantes sur les composants : la caractérisation, la classification, la recherche, la sélection, l'adaptation, l'agrégation, le test et l'évaluation.

Les travaux de caractérisation se sont davantage concentrés jusqu'à maintenant sur les attributs de déploiement du composant [11] et celle des services fournis et des services requis [20]. Il n'y a pas de métadonnées qui regroupent également les attributs statiques, techniques, d'agrégation, des plateformes, métiers et de qualité. Quant à la classification, Tochiano et Jaccheri proposent une classification basé sur les attributs techniques [4]. Ils se basent sur dix attributs regroupés dans quatre groupes soient : l'origine du composant, sa personnalisation, son paquetage compilé et son rôle. Cette classification ne prend pas en considération les aspects d'intégration tels que le niveau et le potentiel de couplage, le degré d'accessibilité, la couche logicielle du composant (présentation, traitement, données) et la méthode d'agrégation (connecteur spécifique, web service, etc.). Ces attributs sont utiles pour décider de la faisabilité d'agrégation.

Rares sont les travaux qui composent les paradigmes du composant logiciel et d'ontologie. En fait, une ontologie est une forme de représentation sémantique des métadonnées pour la description enrichie des ressources [5], notamment des composants logiciels. La description des composants logiciels par des ontologies est une nouvelle approche qui permet de partager les connaissances sur les composants et leur contribution potentielle dans la construction de systèmes à base de composant. IZZA traite les composants logiciels de point de vue service. Il propose une approche flexible basée sur les services sémantiques, en combinant à la fois les ontologies et les Services Web [3]. Mikhail déduit que les intégrations des applications actuelles se concentrent essentiellement sur les aspects syntaxiques des interfaces applicatives [10]. Il rappelle que les aspects sémantiques, qui décrivent le domaine de l'application, manquent. Par conséquent, on se trouve avec des intégrations syntaxiquement réalisables, mais ne répondent pas nécessairement aux besoins de l'intégration. Ainsi le besoin d'intégrer la logique des composants dans leur description s'impose afin de réussir une agrégation aussi bien syntaxiquement que sémantiquement.

Dans la suite, nous présentons la structure SOCOM qui décrit les composants logiciels prenant en compte différents aspects des métadonnées des composants ainsi que le volet sémantique.

3. LA STRUCTURE DES COMPOSANTS LOGICIELS SOCOM

Nous avons développé antérieurement la structure de métadonnées des composants logiciels SOCOM. Comme nous l'avons défini auparavant, cette structure couvre quatre aspects d'un composant logiciel (technique, non fonctionnel, agrégation, métier). Nous avons raffiné cette structure par l'ajout de deux types d'attributs (statique et plateformes). Nous avons remplacé le terme non fonctionnel par qualité et le terme technique par service. Nous voulons par ces changements enrichir notre modèle abstrait et indépendant de la plateforme des composants et nous conformer le plus possible à la terminologie du domaine. D'une façon plus précise, à partir de notre modèle, nous voulons être capables de décrire de façon complète plusieurs types de composants. Aussi, nous espérons générer les spécifications fonctionnelles et techniques de ce même composant dans son langage d'implémentation. Par exemple, les services d'un composant développé en Java seront décrits par instanciation de notre modèle. Lors d'un processus d'agrégation de ce composant avec un autre développé aussi en Java, nous voulons générer automatiquement les APIs[4] des deux composants et les réutiliser dans un scénario d'agrégation spécifique. En résumé, pour chaque composant, on distingue les éléments des six paragraphes suivants.

Les ***attributs statiques*** décrivent les caractéristiques statiques d'un composant tels que son nom logique, sa description générale, son identificateur local et global, etc. Une fois créés, ces attributs sont presque stables et donnent des informations générales sur le composant logiciel. Ce type d'attributs est dédié à des acteurs humains et systèmes qui ne s'intéressent pas à la connaissance du comportement technique détaillé et de l'environnement du composant.

Les ***attributs des services*** décrivent le comportement interne et externe du composant. En effet, ils décrivent d'une part les services offerts par le composant pour servir son environnement externe, soit le contexte où il est développé. D'autre part, ils décrivent les services requis par le composant pour assurer son fonctionnement interne et assumer ses responsabilités. D'autres attributs du composant sont couverts par notre structure comme l'identificateur local et global du composant, le nom du composant, et l'emplacement de son paquetage technique.

Les ***attributs de qualité*** sont des aspects dits non techniques (aussi nommés attributs non fonctionnels dans la communauté d'architecture logicielle) du composant mais qui ont un impact sur une réutilisation potentielle de ce composant dans un domaine autre que celui de sa création. Nous traitons des aspects tels la fiabilité, la sécurité, l'extensibilité, le ou les droits d'utilisation, la version, la licence, la performance, l'intégrité, etc.

Les ***attributs des plateformes*** couvrent certains aspects non fonctionnels de l'environnement du composant au niveau des plates-formes de développement, de configuration, de modélisation, de la persistance (données) et des langages de programmation. Nous avons ajouté aussi des attributs de déploiement afin d'automatiser ultérieurement le déploiement du

[4] API (Application Programming Interface): Interface des programmes en Java

composant une fois le processus d'agrégation stabilisé. Ce volet ne sera pas discuté ici car il dépasse le cadre de cet article, demandant trop de développement vu son importance dans la faisabilité des agrégations et le déploiement des agrégats.

Les ***attributs d'agrégation***, aussi nommés attributs de composition, visent à capter des informations sur le contexte d'agrégation du composant. On note les attributs concernant le degré d'accessibilité, le niveau de couplage, la couche logicielle à laquelle il appartient, son modèle de composant UML et la catégorie de l'agrégation du composant.

Les ***attributs d'affaires***, aussi nommé attributs métiers. Ils décrivent les fonctionnalités des composants. Il s'agit des cas d'utilisation qui rappellent les services du composant de haut niveau en langage naturel. Ces informations sont utiles pour s'informer et analyser la faisabilité d'agrégations des composants logiciels.

On rappelle que les métadonnées de composant SOCOM sont détaillées dans les articles suivants [7], [8] et [9].

4. LE MODÈLE ONTOLOGIQUE DANS MOT+OWL

Avant de parler du modèle de l'ontologie de SOCOM, nous rappelons la définition d'une ontologie. Une ontologie est un vocabulaire commun que se donnent des usagers ayant besoin de partager des informations dans un domaine. L'ontologie regroupe des définitions lisibles en machine au moyen des concepts de base de ce domaine exprimés sous forme de classes, des relations entre ces classes et d'axiomes énonçant des propriétés de ces classes et de ces relations [19]. Dans notre cas, le domaine étudié est le développement des logiciels à base de composants et les informations que nous voulons partager sont les différents concepts de métadonnées des composants, leurs relations et leurs propriétés.

Pour comprendre le modèle OWL de SOCOM, nous expliquons rapidement quelques formes et notations graphiques des modèles ontologiques de l'outil MOT+OWL que nous avons utilisé. À l'origine, l'éditeur graphique MOT, développé au LICEF [15][16] est conçu pour concevoir des modèles pédagogiques et des modèles de connaissances de toutes sortes [13]. Récemment, l'éditeur a été spécialisé en un outil MOT+OWL [12] permettant de construire graphiquement des ontologies selon le standard OWL (Ontology Web Language).

Cet éditeur utilise trois formes graphiques mises en relation par des liens typés. Les rectangles représentent les concepts ou classes, les hexagones représentent les propriétés entre les relations et les rectangles aux coins coupés représentent les individus ou instances (voir l'exemple Figure 1). Divers types de liens sont utilisés entre ces trois types d'entité et des étiquettes sur les objets représentent les axiomes. Les types de liens incluent : le lien R entre une propriété et sa classe d'origine (domaine) ou cible (co-domaine), le lien S entre une classe et ses sous-classes, le lien DISJ indiquant que deux classes sont disjointes, le lien I entre une classe une instance qui en fait partie, et un axiome indiquant qu'au moins deux individus du co-domaine sont associés par la propriété à chaque individu du domaine

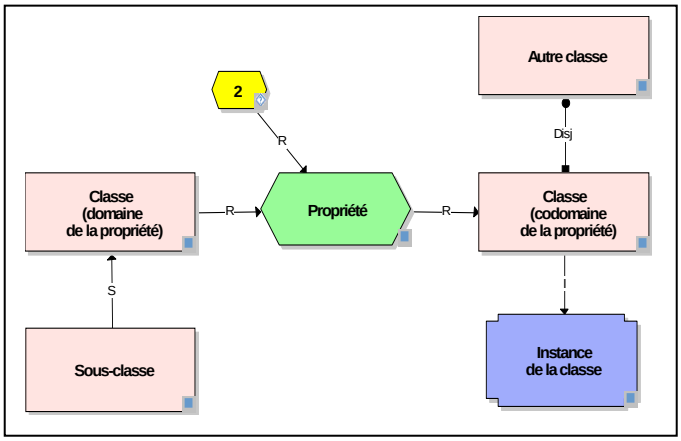


Figure 1. Les formes graphiques des objets et de certains lien dans MOT+OWL

5. LE MODÈLE ONTOLOGIQUE DE LA STRUCTURE SOCOM

Pour le développement du modèle de l'ontologie de SOCOM, nous avons trouvé beaucoup d'outils tels que SWOOP, Protégé[5], WebOnto[6] , etc. Quelques uns offrent la modélisation graphique utilisant différentes formes géométriques, mais nous avons choisi MOT[7] +OWL[8] parce qu'il offre un formalisme graphique complet, simple et expressif pour la représentation des ontologies qui facilite l'activité de conception. Nous avons aussi utilisé l'outil Protégé à cause de ses interfaces usagers qui facilite notamment le peuplement de l'ontologie par des instances de classe. Nous ajoutons qu'il est bien documenté, soutenu et utilisé par une large communauté. Aussi Protégé offre la connexion via HTTP aux moteurs d'inférences pour fournir des déductions et tester aussi bien la consistance que la validité de nos fichiers OWL. Avec cette connexion, Protégé transforme l'ontologie en format DIG, ce qui nous permet d'interroger l'ontologie à l'aide des requêtes qui utilisent le concept « asks ». En plus, l'utilisation de ces deux outils nous permet de gérer des ontologies de bout en bout, depuis la conception initiale jusqu'au raisonnement. Ceci justifie notre choix pour réaliser les travaux dans cet article avec des outils MOT+OWL et Protégé. Deux moteurs d'inférence compatibles avec les spécifications DIG et qui s'appuient sur la

JFO 2008, December 1-3, 2008, Lyon, France.

[5] Protege : Un éditeur d'ontologie en OWL, http://protege.stanford.edu/

[6] WebOnto : Un éditeur d'ontologie Web http://kmi.open.ac.uk/projects/webonto/

[7] OWL : Un langage d'ontologie web, http://www.w3.org/2004/OWL/

[8] OWL : Un langage d'ontologie web, http://www.w3.org/2004/OWL/

logique descriptive sont aussi utilisés, à savoir Pellet[9] et Fact++[10]. Tous les détails sur notre utilisation de ces outils et des spécifications DIG sont explicités dans les sections 5. et 6.

5.1 Le modèle MOT+OWL de l'ontologie de SOCOM

La Figure 2 montre le modèle de premier niveau de l'ontologie SOCOM. En fait, ce modèle reprend la description textuelle de la section 3. Le modèle de l'ontologie de SOCOM montre que le concept « SoftwareComponent » possède des propriétés telles que: « hasStaticSpecification », « hasServiceSpecification », « hasPlatformSpecification », « hasAggregationSpecification », « hasQualitySpecification », « hasBusinessSpecification », qui le lient par la relation R respectivement aux concepts « StaticSpecification », « ServiceSpecification », « PlatformSpecification », « AggregationSpecification », « QualitySpecification » et « BusinessSpecification ». Il ajoute le concept (aussi appelé Classe) agrégat « aggregate », qui se compose, au minimum, d'un composant logiciel. L'agrégat est lui aussi un composant logiciel et par conséquent, il est lié par un lien S au concept « SoftwareComponent ».

Nous avons développé en MOT+OWL notre modèle ontologique de SOCOM, et nous explorons dans ce qui suit le sous-modèle de certains concepts et nous détaillons parfois le contenu par d'autres sous-modèles.

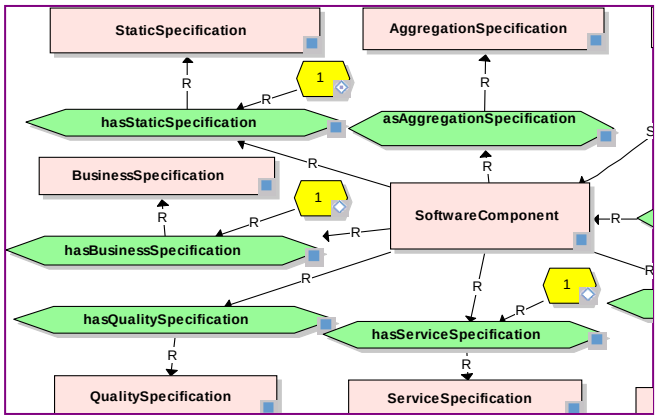


Figure 2. Le modèle ontologique de SOCOM (niveau 1)

5.1.1 Modèle ontologique des spécifications de service «ServiceSpecification »

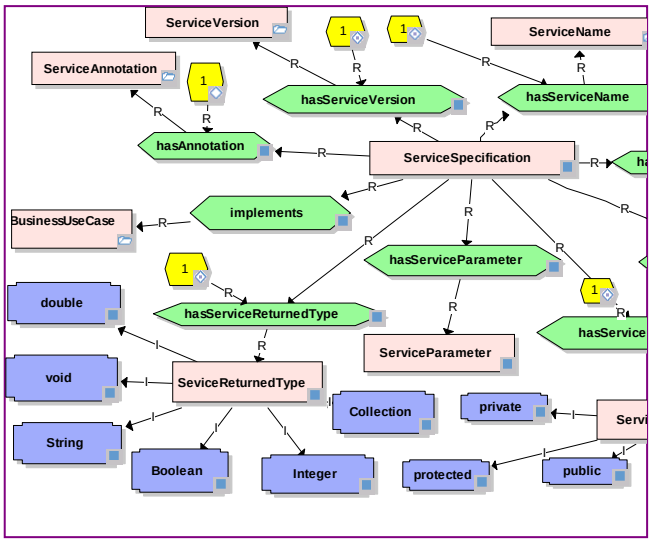


Figure 3. Le modèle ontologique des services du composant SOCOM (niveau 2)

La Figure 3 montre les différents concepts qui décrivent un service fourni ou requis par un composant. Ces concepts traduisent les attributs de service tels que le nom, le type (requis, fourni), le type de retour, la version, le mode d'invocation, la visibilité et les paramètres nécessaires pour son invocation. Pour chacun de ses paramètres, nous décrivons son nom, son type, son ordre dans la liste des paramètres, sa visibilité et son type d'accès (Voir Figure 4)

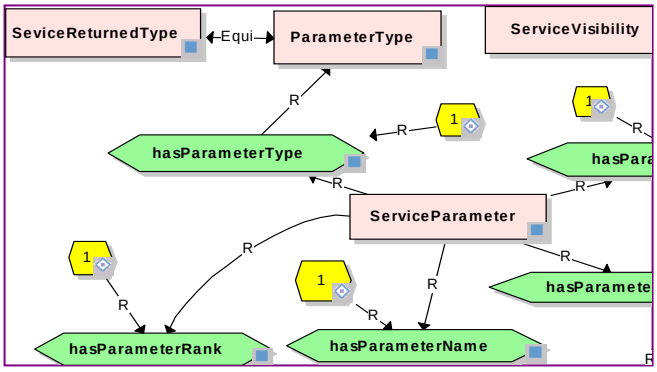


Figure 4. Le modèle ontologique des paramètres des services des composants SOCOM (niveau 3)

On précise que le concept service est relié au concept des spécifications d'affaires « BusinessUseCase » par la relation « implements». Le concept « BusinessUseCase » est une référence du sous modèle de l'ontologie des spécifications d'affaires « BusinessSpecification ». Cette relation est importante puisqu'elle nous permet d'ajouter une sémantique par dessus la description syntaxique du service d'un composant, soit le cas d'utilisation d'affaire qu'il implémente. Moyennant cette relation, on peut avoir une connaissance de haut niveau sur le ou les cas d'affaire réalisés par le service du composant en question. Aussi, cette

[9] Pellet : Un moteur d'inférence en Java, http://www.mindswap.org/2003/pellet/

[10] Fact++ : Un moteur d'inférence basé en C++, http://owl.man.ac.uk/factplusplus/

sémantique sur le service est utile pour un analyste qui recherche un composant particulier pour des fins d'agrégations avec d'autres. Ceci nous aligne avec le paradigme de composant logique détaillé dans les travaux de Renaux et al. [17].

5.1.2 Modèle ontologique des specifications d'affaire «BusinessSpecification »

La Figure 5 montre les spécifications d'affaire (métier) du composant SOCOM. En effet, un composant réalise à haut niveau un ou plusieurs cas d'utilisation d'affaire. Pour chaque cas d'affaire, on associe une description et un acteur. Ce dernier peut être humain ou système et il est responsable de la réalisation du cas en question. Par ce concept, la description du composant SOCOM dispose d'une double description pour chaque service, soient des informations techniques fournies par le concept « ServiceSpecification » et celles des affaires fournies par « BusinessSpecification ».

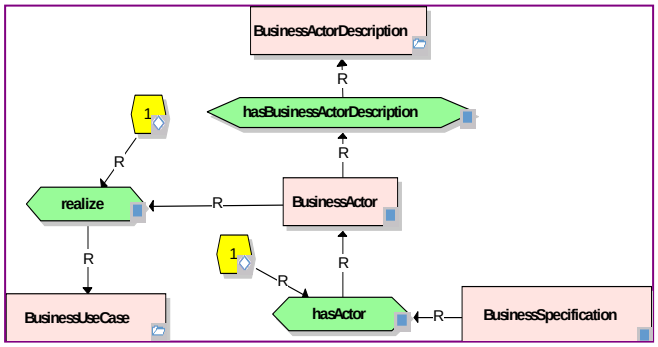


Figure 5. Le modèle ontologique des spécifications d'affaire de SOCOM (niveau 2)

L'ontologie des spécifications d'agrégation est un élément clef pour l'étude de faisabilité des agrégations des composants logiciels. En fait, les concepts de la figure 6 tels que degré d'accessibilité, niveau de couplage, potentiel de couplage, couche logicielle, type du langage du composant et méthode d'agrégation du composant sont utilisés pour classer le composant dans un sous ensemble homogène de composants. Nous avons proposé une classification qui utilise un sous ensemble de ses propriétés dans [7]. Cette homogénéité traduit la possibilité d'agrégation des composants appartenant à un même sous ensemble. En d'autres termes, si deux composants possèdent un ensemble de propriétés communes, ils se trouveront dans un même sous ensemble. On peut déduire qu'ils sont deux composants candidats pouvant participer dans un processus d'agrégation.

Dans ce qui suit, on présente le sous modèle de l'ontologie des spécifications des plateformes. Le composant passe par différents types de plateformes depuis son développement à son exploitation tels que système, modélisation, développement, déploiement, de données, etc. L'ontologie des plateformes nous classe les composants dans des sous-ensembles homogènes par type de plateforme. Ceci peut nous aider dans l'étude de faisabilité des différentes catégories d'agrégation. La figure 7 ci-dessous montre respectivement cette ontologie des plateformes (2ème niveau) et quelques instances de la classe de la plateforme.

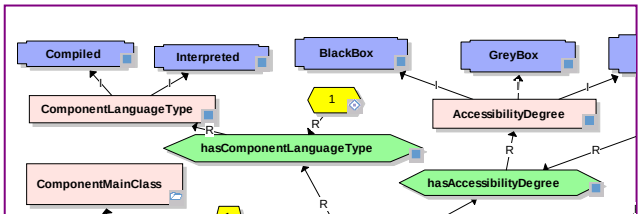


Figure 6. Le modèle ontologique des spécifications d'agrégation de SOCOM (niveau 2).

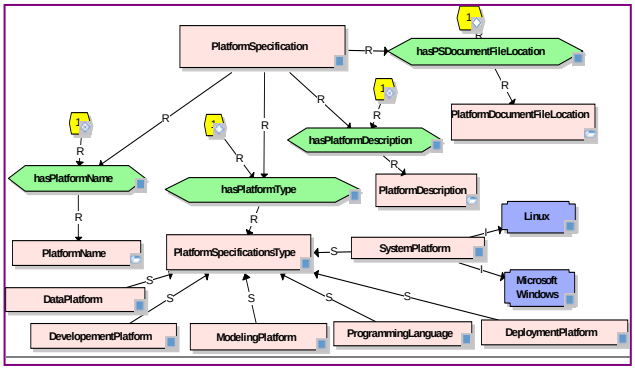


Figure 7. Le sous modèle ontologique des spécifications des plateformes de SOCOM (niveau 2).

5.2 L'ontologie de SOCOM en OWL DL sous Protégé

La Figure 8 montre un aperçu des instances des spécifications d'agrégation d'un composant de la l'ontologie SOCOM décrites dans la section 4. En effet, les classes de l'ontologie SOCOM occupent le coté gauche de la figure. Les instances des classes sont affichées au milieu. À droite, nous retrouvons les valeurs des propriétés de l'instance sélectionnée dans la colonne du milieu. Par exemple, on référence une trentaine de composants de la banque en construction qui sera utilisée dans TELOS

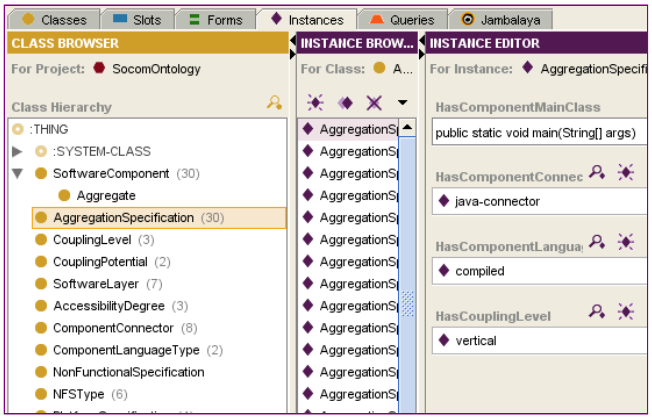


Figure 8. Une partie de l'ontologie SOCOM sous Protégé OWL.

« SoftwareComponent» est une super-classe de « Aggregate ». De ce fait, toutes les instances de la classe « Aggregate » se trouvent avec les propriétés de la classe « SoftwareComponent » parce qu'un agrégat est aussi un composant logiciel. Les propriétés de la classe « Aggregate » sont des propriétés inférées qui sont héritées de la classe « SoftwareComponent ». Aussi, les instances de la classe « Aggregate » se retrouvent dans l'ensemble des instances de la classe « SoftwareComponent ». Elles s'identifient comme des instances inférées de la classe « Aggregate ».

6. IMPLÉMENTATION DE L'ONTOLOGIE SOCOM

6.1 Les logiques descriptives

Nous avons choisi le langage d'ontologie OWL DL comme base pour un certain nombre de raisons. Il s'agit d'un des trois langages d'ontologie pour le Web recommandé par le W3C pour développer le Web sémantique. De ces trois langages, OWL DL est destiné aux utilisateurs qui demandent une expressivité maximale tout en retenant la complétude du calcul (toutes les inférences sont garanties calculables) et la décidabilité (tous les calculs s'achèveront dans un intervalle de temps fini).

Le langage OWL DL (DL étant les initiales pour Description Logic) a été conçu pour correspondre à une des logiques descriptives et fournir ainsi un langage offrant les propriétés de calcul nécessaires à des systèmes de raisonnement. Le logiciel graphique MOT+OWL que nous avons utilisé est fondé sur ce standard. Il permet d'exprimer complètement les primitives du standard OWL DL et de produire le fichier XML correspondant, lequel peut ensuite être réutilisé dans les traitements.

Les pierres d'assise des logiques descriptives sont les concepts, les propriétés et les individus. Les concepts décrivent les propriétés communes d'une collection d'individus. Ils peuvent être considérés comme des prédicats de premier ordre qui sont interprétés comme un ensemble d'objets. Les propriétés sont des relations binaires entre les objets. Des axiomes tels que « à tout individu d'une classe correspond au moins un individu d'une autre » ou « aucun individu n'appartient à la fois à la classe A et à la classe B » peuvent être ajoutés pour décrire un domaine de façon précise et guider un moteur d'inférence dans les déductions qui lui permettent d'exécuter une requête d'un usage. Notre ontologie de SOCOM utilise le langage OWL DL [19]

6.2 Les specifications DIG

Les moteurs d'inférence qui s'appuient sur les logiques descriptives sont de plus en plus utilisés pour raisonner sur des ressources du Web sémantique, c'est-à-dire référencées au moyen d'ontologies. Pour permettre à un client d'interagir avec différents moteurs d'une manière standard, une interface standard et commune est souhaitable. Le groupe DIG[11] œuvre dans la conception des systèmes compatibles avec les logiques descriptives. Une de ses activités consiste à développer une interface normalisée en XML pour les systèmes qui utilisent ces logiques en leur fournissant une API de base. L'interface DIG est une nouvelle norme qui permet l'accès à tout moteur d'inférence compatible via des interfaces Web utilisant le protocole HTTP. Cette interface est supportée par la majorité des moteurs d'inférence et facilite la construction des composants logiciels réutilisables. Des moteurs d'inférences comme Pellet, Racer et Fact++ sont compatibles avec les spécifications DIG.

Ces spécifications se composent essentiellement d'un schéma XML décrivant le formalisme du langage descriptif. Deux opérations sont disponibles pour construire et questionner une ontologie. L'ontologie est construite par l'opération « tells » et interrogée par l'opération « asks ». Pour plus de détails sur les spécifications DIG de version 1.1, voir [2]

6.3 Un exemple de requête et de réponses DIG

Pour expliquer la logique des requêtes DIG, nous présentons les deux structures DIG les plus utilisées, soient « asks » et « responses ». Le script de la Figure 9 illustre une requête « asks » de type « relatedIndividuals » qui interroge l'ontologie présentée plus haut pour demander les individus des concepts qui sont liés à la propriété «hasServiceSpecification ». Autrement dit on cherche la liste des composants et leurs services intégrés comme des instances dans l'ontologie. Dans la requête, il faut spécifier l'identificateur de la requête «qTelosComponentServices » et la propriété « hasServiceSpecification » qui rattache les concepts « SoftwareComponent» et «ServiceSpecifcation».

La réponse à cette requête est le fichier XML de la Figure 10. Ce dernier est retourné par le moteur d'inférence et montre les résultats trouvés sous la forme d'individus en langage ontologique ou bien les instances en langage objet. La liste retournée est un ensemble de paires. En effet, le premier individu de la paire est l'instance du concept source « SoftwareComponent » alors que le deuxième est l'instance du concept « ServiceSpecification». Le tout est encapsulé dans la balise « responses ». Cette réponse doit rappeler l'identificateur de la requête. On note que « TreeEditor » et « CPEModel » sont respectivement les noms des composants « Éditeur des activités » et « Content Properties Editor Model ». Le premier permet la construction de la structure arborescente des activités d'un cours ou d'un processus de travail et le deuxième permet l'édition du contenu des propriétés des activités de l'arborescence précédemment construite.

```
<asks xmlns="http://dl.kr.org/dig/2003/02/lang">
   <relatedIndividuals id="qTelosComponentServices">
        <ratom name="hasServices"/>
   </relatedIndividuals>
</asks>
```

Figure 9. Exemple d'une requête DIG utilisant le concept « asks ».

```
<responses xmlns="http://dl.kr.org/dig/2003/02/lang">
  <individualPairSet id="qTelosComponentServices">
    <individualPair>
      <individual name="CPEModel"/>
      <individual name="getCPEModelService"/>
    </individualPair>
    <individualPair>
      <individual name="TreeEditor"/>
      <individual name="performTreeEditorService"/>
    </individualPair>…</individualPairSet>
</responses>
```

Figure 10. Exemple d'une réponse DIG utilisant le concept « responses ».

[11] DIG: http://dl.kr.org/dig/

7. IMPLÉMENTATION DE L'ONOTLOIE DE SOCOM DANS UNE ARCHITECTURE DISTRIBUÉE

Techniquement, la structure et les données de l'ontologie de SOCOM sont stockées dans un fichier OWL. Plusieurs interfaces de programmation en Java et en C++ sont disponibles pour interroger des fichiers OWL (API OWL de Protégé[12], Jena API[13], etc.). Cependant, nous ne voulons pas nous restreindre à une implémentation spécifique pour interroger notre ontologie, ce qui nous a conduit à utiliser les spécifications DIG (voir section 6.2). À cet effet, nous avons développé un client Java qui utilise une API Java des spécifications DIG [2]. Nous avons transformé l'ontologie du format OWL en format DIG basé sur l'opération « tells ». Notre client utilise le fichier XML des requêtes basées sur l'opération « asks » et l'ontologie traduite en un fichier XML construit avec le concept « tells ». Ces deux fichiers sont transmis par HTTP à un moteur d'inférence compatible DIG (Pellet dans notre cas). Ce dernier exécute la requête et retourne une réponse sous forme d'un fichier XML encapsulé dans le concept « responses ». Les bibliothèques Java des spécifications DIG sérialisent la réponse XML en objet Java « DocumentResponse ». En utilisant cet objet, le client Java extrait les informations nécessaires pour les afficher à l'usager ou les utiliser dans un processus d'agrégation des composants logiciels.

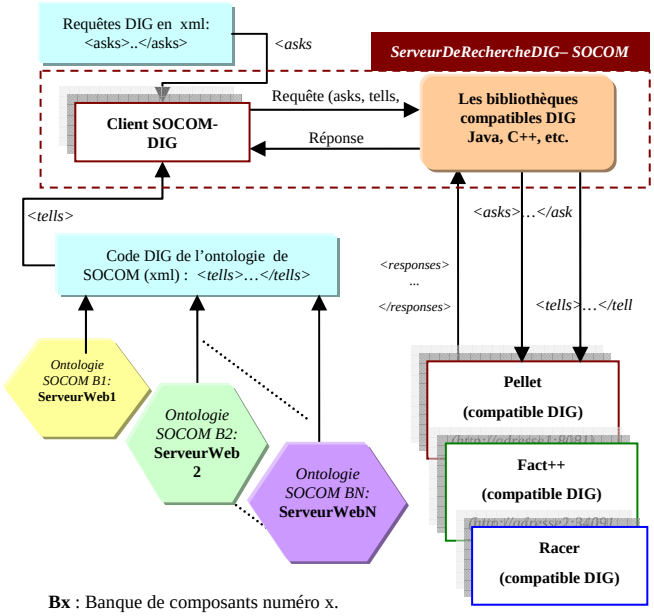
Pour ce qui est de l'architecture distribuée, les ontologies des composants logiciels qui implémentent la structure SOCOM peuvent être déployées sous différents serveurs Web. Pour assurer leur gestion en termes d'exploration et d'instanciation des concepts, un administrateur peut utiliser un outil comme Protégé, WebOnto ou Swoop[14]. Le modèle de la Figure 11 montre un scénario de recherche sur les ontologies géographiquement distribuées qui référencent les composants logiciels. Autrement dit, nous pouvons distribuer sur différents serveurs les ontologies de SOCOM, les requêtes DIG, les moteurs d'inférences et le code de traitement qui implémente les interfaces DIG en plusieurs langages. (Voir le serveur de recherche DIG –SOCOM dans la Figure 11).

Les avantages d'une telle architecture sont basés sur les requêtes DIG et la description DIG, par l'opération « tells » des ontologies de SOCOM. Celles-ci sont indépendantes des plateformes d'inférence et du langage d'implémentation du client. Aussi, nous ne sommes pas contraints d'utiliser un moteur d'inférence spécifique. Tous les moteurs qui sont compatibles DIG peuvent répondre à nos requêtes. Nos requêtes et nos réponses utilisent un format XML compatible DIG, par conséquent nous pouvons créer n'importe quel type de client dans n'importe quel langage de programmation dés qu'il permet la manipulation des requêtes et des réponses DIG dans son code natif en termes de sérialisation et de désérialisation.

[12] API OWL de Protege : http://protege.stanford.edu/plugins/owl/api/

[13] API de Jena : http://jena.sourceforge.net/

[14] Swoop: Editeur OWL, www.mindswap.org/2004/SWOOP/



Bx : Banque de composants numéro x.

@MI : adresse web du moteur d'inférence

Figure 11. L'architecture distribuée des banques des composants(SOCOM, DIG)

8. Une comparaison entre deux implémentations des spécifications SOCOM : Base de données relationnelle vs ontologie

Nous avons aussi implémenté les spécifications SOCOM sous forme d'une base de données relationnelle sous MySQL[15]. Nous avons testé la manipulation des données de SOCOM dans les deux cas et nous avons remarqué une simplicité et une complexité pour chacun des modèles relationnel et ontologique. Nous traitons l'énoncé suivant :

« Nous recherchons un composant type boite noire qui a une licence 'LGPL'. Il doit être implémenté dans le langage Java avec un faible potentiel de couplage. Ce composant doit fournir un service qui s'intitule 'getLOMResources' ». En langage SQL, cette recherche doit être formulée à l'aide de quatre requêtes appelées successivement comme suit :

```
R1: <SELECT componentId, componentName FROM socom_technical
WHERE componentCouplingLevel = "loosely" AND languageType =
"Java">

R2: <SELECT componentId FROM socom_nonfunctional WHERE
licenseName = "LGPL" AND nonFunctionalType = "Licensing" AND
componentId="cptID1">

R3: <SELECT serviceId FROM component_services WHERE serviceName
= "getLOMResources" And componentId="cptId1">

R4: <SELECT * FROM service_parameters WHERE serviceId = "Id-
2121-Service" AND componentId="cptId1">
```

Figure 12. La liste des requêtes SQL qui réponde au besoin de l'exemple.

[15] MySQL : Système de gestion de bases de données relationnelles (www.mysql.org)

Après l'exécution de chaque sous requête, nous devons récupérer la valeur de la variable « componentId » pour l'utiliser dans la suivante comme critère de sélection. Ce traitement additionnel doit se faire à l'aide d'un langage hôte tel que Java qui utilise un connecteur spécifique à notre base de données.

Pour formuler la même requête en DIG, nous utilisons les deux sous concepts du concept « asks »: le concept « instances », pour chercher l'instance du concept, et le concept « stringequals », qui retourne les individus ayant les propriétés recherchées. En effet, la première requête sera appliquée au concept « SoftwareComponents », ce qui traduit notre besoin de recherche d'un composant qui satisfait les critères énoncés dans la deuxième requête. Celle-ci indiquera les relations qui doivent être validées avec les valeurs des instances des classes cibles. Dans notre cas, les relations concernées (propriétés) et les instances (individus) dans l'ordre sont : (« hasLicenseType », LGPL), (« hasAccessibilityDegree », BlackBox), (« hasLanguageType », Java), (« hasCouplingLevel », loosely) et (« hasServiceName », getLOMResources). La syntaxe est expliquée dans le document des spécifications DIG 1.1

```
<asks xmlns="http://dl.kr.org/dig/2003/02/lang">
<instances id="queryGetSpecificComponent">
           <stringequals val="LGPL">
               <ratom name=" hasLicenseType"/>
           </stringequals>
           <stringequals val="BlackBox">
               <ratom name=" hasAccessibilityDegree"/>
           </stringequals>
           <stringequals val="Java">
               <ratom name=" hasLanguageType"/>
           </stringequals>
           <stringequals val="loosely">
               <ratom name="hasCouplingLevel "/>
           </stringequals>
           <stringequals val="getLOMResources">
               <ratom name="hasServiceName "/>
           </stringequals></instances> </asks>
```

Figure 13. La requête DIG pour faire la recherche du composant de l'énoncé.

On peut constater que nous avons construit la requête (Figure 13) en traduisant simplement la description de l'énoncé ci-dessus. Cette simplicité dans la formulation de la requête DIG découle de l'expressivité de la syntaxe des requêtes DIG et de la classification des concepts et des relations de l'ontologie technique de SOCOM. En effet, cette syntaxe englobe mieux la sémantique de l'énoncé de l'exemple. En plus, les résultats de recherches peuvent alimenter l'ontologie avec le concept « tells » pour enrichir l'ontologie des composants SOCOM avec de nouvelles connaissances. En d'autres termes, la force du modèle ontologique réside dans sa structure qui est conforme à la logique descriptive et dans son couplage avec les moteurs d'inférence pour servir du raisonnement via des requêtes intelligentes. Toutefois, il est important de rappeler que notre logique de recherche est partagée entre l'ontologie technique de SOCOM et le langage de requête DIG.

Ainsi, on note un des avantages des ontologies vis-à-vis des bases de données, soit l'expressivité et la simplicité d'implémenter les requêtes depuis une recherche exprimée en langage naturel. À ce niveau, on parle de deux dimensions de la comparaison soient, la structure conceptuelle et le langage de requête des bases de données vis-à-vis de celles des ontologies. Cette simplicité pourrait être obtenue avec une base de données, mais au prix du développement d'un programme Java ou autre qui implémente les structures des propriétés et des axiomes exprimés dans l'ontologie technique de SOCOM.

Les individus de l'ontologie SOCOM sont stockés dans un fichier OWL conforme à une ontologie technique épousant le standard OWL-DL. On rappelle que les individus de l'ontologie sont équivalents aux données d'une base de données. Aussi, on appelle la structure et les individus de l'ontologie de SOCOM respectivement ontologie technique et ontologie d'instances. Elles sont sauvegardées dans deux fichiers OWL distincts. Toutefois, l'ontologie des instances référence l'ontologie technique pour valider la structure des données. Les deux fichiers OWL sont hébergés physiquement sous un serveur Web. Pour accéder à ces deux ontologies, il suffit de connaitre leur adresse web.

La base de données de SOCOM est implémentée dans un SGBDR[16]. Pour y accéder du point de vue applicatif, il faut un programme qui a une connexion (JDBC ou autre) à une base de données avec les droits associés. Il est évident que pour réutiliser cette base de données dans un autre contexte indépendant, il faut avoir un SGBDR équivalent et les scripts de création de la structure et des données pour reconstruire l'environnement des données et pouvoir l'exploiter. Pour ce qui est de la réutilisation de l'ontologie, il faut connaitre seulement l'adresse de l'ontologie technique. N'importe lequel des outils d'ontologie peut pointer sur cette ontologie technique et permettra à son utilisateur de peupler l'ontologie des instances. Et pour une réutilisation dans un réseau privé, il suffit de copier l'ontologie technique et celle des instances dans un serveur Web hébergé dans le réseau en question. Ainsi, on note que les ontologies sont plus avantageuses dans les dimensions : portabilité dans différentes plateformes et simplicité de réutilisation.

Quant à l'évolution, il est aujourd'hui difficile de gérer les changements suite à l'évolution d'une ontologie. En pratique, lorsqu'on a changé notre ontologie technique, nous avons trouvé beaucoup de difficultés pour mettre à jour l'ontologie des instances. On n'a pas pu réutiliser les instances de l'ancienne ontologie technique. De ce fait, on a utilisé l'outil Protégé pour peupler de nouveau, à la main, l'ontologie des instances conformément à la nouvelle ontologie technique. Avec le SGBDR MySQL, le changement des attributs des tables de la base de données de SOCOM est plus facile. Pour remédier à ce problème, on a synchronisé les deux implémentations ontologique et relationnelle par un convertisseur de donnée. Ce dernier traduit les données depuis la base de données SOCOM en fichier OWL qui représente l'ontologie des instances, tout en gardant la conformité avec l'ontologie technique SOCOM. En fait, il y a encore peu de travaux qui traitent de l'évolution des ontologies; on note la proposition d'une méthode et d'outils pour maintenir la consistance du référence des ressources, lors de l'évolution d'une

[16] SGBDR : Système de Gestion de Base de Données Relationnel.

ontologie (Rogozan & Paquette, 2005). En attendant la conclusion de ces travaux, il nous faut noter un avantage des bases de données relationnelles vis-à-vis des ontologies en ce qui concerne la dimension de la gestion du changement (évolution) des métadonnées des composants SOCOM. On conclue que les dimensions d'évolution et de gestion des changements sont pour le moment plus simples à gérer avec les bases de données qu'avec les ontologies.

En exécution, le problème de performance en présence d'ontologies de grande taille et le temps de réponse des moteurs d'inférence vis-à-vis des requêtes complexes demeurent les inconvénients majeurs d'une telle approche. Notre expérience nous montre une simplicité de développement de l'application Java avec une base de données relationnelle sous le SGBDR MySQL et un meilleur temps de réponse pour des requêtes simples. Par contre, l'exemple précédent nous montre que l'équivalent d'une requête simple DIG est une structure complexe lorsque réalisée en SQL. En effet, le langage SQL qui interroge la base de données SOCOM nous fournit peu de sémantique sur les composants logiciels, leur classification et la classification de leurs agrégations possibles. Les modèles ontologiques de ces classifications ont fait l'objet d'un autre article (Masmoudi, Paquette, & Champagne, 2005) et seront développés davantage dans nos prochains travaux.

En résumé, le tableau ci-dessous offre une comparaison de l'implémentation de SOCOM sous la forme d'une base de données et sous la forme d'une ontologie. On rappelle que ces implémentations ont pour objectif d'aider à la recherche des composants logiciels dans une banque de composants décrit avec la structure SOCOM.

Table 1. Tableau comparatif des ontologies vs base de données.

Dimensions / Implémentation de SOCOM	***Base de données***	***Ontologie***
Portabilité sur différents plateforme de stockage	+/-	+
Évolution (Gestion du changement)	+	-
Simplicité de stockage	-	+
Performance	+	-
Expressivité du Langage de requête	-	+
Simplicité du langage de requête	-	+
Gestion des droits d'accès et privilèges d'accès	+	-
Simplicité de réutilisation	-	+

Tous les résultats de la comparaison présentés ci-dessus montrent des avantages et des inconvénients des deux implémentations. Cependant, les avantages de l'implémentation ontologique auraient pu être obtenus par l'implémentation relationnelle sauf qu'il nous aurait fallu développer toute la logique descriptive que les moteurs d'inférences nous offrent. Par conséquent, nous adoptons une solution mixte. En effet, cette solution utilise conjointement le modèle ontologique et relationnel des métadonnées SOCOM. Un convertisseur de données a été développé pour gérer l'évolution de la structure des métadonnées des composants ainsi que pour la synchronisation des données entre la base de données et les instances de l'ontologie.

9. CONCLUSION ET PERSPECTIVE

Cet article présente l'évolution de nos travaux sur l'indexation des composants logiciels et les méthodes de leur agrégation. Comme nous l'avons mentionné dans la section des travaux connexes, une description détaillée et complète des composants logiciels est manquante. De ce fait, les travaux de cet article viennent traiter les aspects du CBSD suivants : la caractérisation, la classification et la recherche des composants logiciels pour des fins d'agrégation. En fait, nous avons commencé par la définition de la structure SOCOM pour décrire les composants logiciels en définissant des spécifications statiques, des services, de qualité, des plateformes, d'agrégation et métiers des composants logiciels. Nous avons développé notre modèle ontologique de SOCOM à l'aide de MOT+OWL. Nous avons utilisé Protégé pour peupler et gérer notre ontologie. Pour promouvoir la recherche des composants logiciels, nous avons adopté une solution mixte dans laquelle nous avons défini un modèle ontologique de SOCOM synchronisé avec un modèle relationnel moyennant un convertisseur de données que nous avons développé. Nous avons justifié ce choix mixte par une comparaison détaillée entre une implémentation en base de données relationnelle et une implémentation ontologique. Cette comparaison a pris en considération un ensemble de dimensions détaillées dans la section 8.

Nous avons profité aussi de l'apport des spécifications DIG vu qu'elles nous permettent de formuler des requêtes complexes en XML indépendantes des moteurs d'inférences compatibles DIG.

Dans nos travaux futurs, nous essaierons de définir et d'intégrer des règles d'agrégation. Ceci vient compléter un autre aspect du CBSD, soit la sélection des composants pour étudier la faisabilité d'agrégation. En effet, nous avons catégorisé les agrégations des composants logiciels en nous appuyant sur trois caractéristiques des composants (Masmoudi, Paquette, & Champagne, 2005). Ultérieurement, nous ajusterons cette catégorisation et nous étudierons l'apport des ontologies pour aider un ingénieur ou un technologue dans le processus d'agrégation de composants logiciels. Ceci aidera à la construction des systèmes d'apprentissages à distance ou de gestion des connaissances par agrégation des composants intégrés dans la banque de ressource d'un système comme TELOSThe heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

10. REMERCIMENTS

Nous tenons à remercier le CRSNG[17] d'avoir financé le projet LORNET et l'équipe du laboratoire LICEF qui développe actuellement le système TELOS. Nous remercions les réviseurs anonymes d'avoir contribué à l'amélioration de cet article.

[17] CRSNG : Conseil de recherches en Sciences Naturelles et en Génie du Canada

11. BIBLIOGRAPHIES

[1] Allen, P., & Frosts, S. (Mars 1998). Tackling the Key Issues in CBD. In *Select Software Tools*.

[2] Bechhofer, S. (2003). DIG Specifications, version 1.1. from http://dl-web.man.ac.uk/dig/2003/02/interface.pdf

[3] IZZA, S. (2006). *INTEGRATION DES SYSTEMES D'INFORMATION INDUTRIELS, Une approche flexible basée sur les services sémantiques.* Ecole Nationale Supérieure des Mines de Saint-Etienne, Sainte-Étienne.

[4] M. Torchiano, L. Jaccheri, C.-F. Srensens, & A. I. Wang. (2002). *COTS Products Characterization.* Paper presented at the Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy.

[5] M.C. Daconta, L.J. Obrst, & Smith, K. T. (2003). *The Semantic Web, A Guide to the Future of XML, Web Services, and Knowledge Management*.

[6] Magnan, F., & Paquette, G. (2006, June). *TELOS: An ontology driven eLearning OS.* . Paper presented at the Proceeding of the SOA-AIS-2006 Worskhop, Dublin, Ireland.

[7] Masmoudi, A., Paquette, G., & Champagne, R. (2005). *Agrégation de Composant Dirigée par les Métadonnées.* Paper presented at the INFORSID, Grenoble-France.

[8] Masmoudi, A., Paquette, G., & Champagne, R. (2006). *Metadata-Based software components repository's Reuse.* Paper presented at the I2LOR'06, Montreal-Canada.

[9] Masmoudi, A., Paquette, G., & Champagne, R. (2008). Metadata-driven software components aggregation process with reuse. *International Journal Advanced Medi and Communication*, p 35-58.

[10] Mikhail, K. (2004). *A methodology of semi-automated software integration: an approach based on logical inference.* Unpublished Computer Science, INSA de rouen.

[11] Mili, H., Mili, A., Yacoub, S., & Addy, E. (2002). *Reuse based software engineering : techniques, organization, and measurement*. New York: Wiley.

[12] Paquette G. (2007) Graphical Ontology Modeling Language for Learning Environments, Technology, Instruction., Cognition and Learning , Vol.5 , p.133-168, Old City Publishing, Inc.

[13] Paquette, G., Léonard, M., Lundgren-Cayrol, K., Mihaila, S., & Gareau, D. (2005). Learning Design based on Graphical Knowledge-Modeling *Journal of Educational technology and Society ET&S , Special issue on Learning Design*(Proceedings of the UNFOLD-PROLEARN Joint Workshop, Valkenburh, The Netherlands, September 2005 on Current Research on IMS Learning Design, 2006).

[14] Paquette, G., Rosca, I., Mihaila, S., & Masmoudi, A. (2006). TELOS, a service-oriented framework to support learning and knowledge Management In S. Pierre (Ed.), *E-Learning Networked Environments and Architectures: a Knowledge Processing Perspective*: Springer-Verlag.

[15] Paquette, Gilbert (2002) *Modélisation des connaissances et des compétences, pour concevoir et apprendre*, 357 pp, Presses de l'Université du Québec.

[16] Paquette, Gilbert (1996) La modélisation par objets typés: une méthode de représentation pour les systèmes d'apprentissage et d'aide a la tâche. *Sciences et techniques éducatives* , France, avril 1996

[17] Renaux, E., Olivier, C., & Jean-Marc, G. (2004, mars). *Chaîne de production de systèmes à base de composants logiques* Paper presented at the LMO 2004.

[18] Rogozan, D., & Paquette, G. (2005, Septembre). *Managing Ontology Changes on the Semantic Web.* Paper presented at the WI-2005 Web Intelligence Conference, France.

[19] Smith, M. K., Welty, C., & McGuinness, D. L. (2004). *OWL Web Ontology Language Guide* (W3C Recommendation).

[20] Szyperski, C., Gruntz, D., & Murer, S. (2002). *Component software : beyond object-oriented programming* (2nd ed.). New York: Addison-Wesley.

[21] Villalobos, J. (2003). *Fédération de Composants : Une Architecture Logicielle pour la Composition par Coordination.* Unpublished Informatique, Université Joseph Fourier, Grenoble.