Implantation d'un moteur de transformation intelligent dans le contexte de l'Architecture Conduite par les Ontologies

Michel Héon

Télé-Université du Québec
100 rue Sherbrooke ouest, Montréal, H2X 3P2
michel.heon@licef.ca

RÉSUMÉ. *L'Architecture Conduite par les Modèles, tel que défini par L' « Object Management Groupe », préconise d'utiliser le « Meta Object Facility » en tant qu'espace de modélisation ainsi que l' « Object Constraint Language » et le « Query / View / Transformation » en tant que langage de transformation entre un modèle source et le modèle cible. Certains principes de l'ACM sont transposés à l'espace de modélisation du web sémantique afin de permettre la transformation d'une ontologie source vers une ontologie cible. Dans le contexte du WS cet article présente une Architecture Conduite par les Ontologies qui utilise l'OWL en tant qu'espace de modélisation et qui implante un de moteur transformation « intelligent » d'une ontologie dont le langage de transformation est le « Semantic Web Rule Language ». Pour produire une ontologie cible à partir d'une ontologie source, les primitives du langage SWRL doivent être étendues afin de permettre la création de classes, de propriétés, d'individus et de restrictions dans l'ontologie cible. De plus, cet article présente les concepts architecturaux qui soutiennent l'implantation des primitives SWRL de construction d'une ontologie cible ainsi que les résultats associés au fonctionnement du moteur de transformation à base de règles SWRL.*

MOTS-CLÉS : *Transformation d'ontologies, Transformation de modèles, Architecture conduite par les modèles, Architecture conduite par les ontologies, moteur de transformation intelligent.*

ABSTRACT. *Model Driven Architecture, as defined by the Object Management Group, advocates using the Meta Object Facility as an modeling space and the Object Constraint Language and Query / View / Transformation as a transformation language between source model and target model. Some principles of MDA are transposed to the modeling space of the semantic web to enable the transformation of a source ontology to a target ontology. In this paper presents a SW Ontology Driven Architecture using OWL as the modeling space and implements a "smart" processing engine of an ontology in which language processing is the Semantic Web Rule Language. To produce a target ontology from an source ontology, the primitive of SWRL should be extended to allow the creation of ontology elements in the target ontology. In addition, this article presents the architectural concepts that support the implementation of SWRL primitive construction of a target ontology and the results associated with the operation of the transformation engine based on SWRL rules.*

KEYWORDS: *Ontology transformation, model transformation, model driven architecture, ontology drives architecture, intelligent processing engine*

1. Introduction

La transformation d'ontologies est l'une des activités au cœur de l'ingénierie ontologique. Cet article présente l'implantation d'une méthode de transformation automatique d'ontologies dans le formalisme de *l'Ontology Web Language*[1] (*OWL*) fondée sur le processus de transformation de modèles tel que défini dans l'Architecture Conduite par le Modèles (ACM) de l'*Object Management Group* (OMG, www.omg.org). Bien qu'étant très efficace pour la transformation de modèles UML, l'architecture conduite par les modèles, ne permet pas de prime abord, la transformation de modèles de type ontologique *à la* OWL. Cependant, il est important de noter qu'en ce domaine, les travaux de Gasévik (2006) ont tout de même produit des résultats qui ont inspirés la présente recherche et que les travaux de Miksa *et al.* (2010) et Staab *et al.* (2010) vont dans le sens que nous avons développé.

Nous présentons une variante ontologique de l'ACM que nous appelons *Architecture Conduite par les Ontologies* (ACO). L'ACO a pour objectif d'offrir un cadre architectural, méthodologique et informatique de transformation d'ontologies de type OWL. Les avantages de l'ACO sont multiples. D'abord elle offre un outil de transformation de modèles pour le domaine du web sémantique. Elle permet aussi d'intégrer au processus de transformation des ontologies les paradigmes de l'approche symbolique de l'IA. Ainsi, le processus de transformation, dit *intelligent,* est enrichi par les mécanismes de raisonnement logique et de raisonnement à base de règles propres aux ontologies et au web sémantique. Finalement, l'implantation informatique de l'ACO se fonde sur les technologies accessibles et issues du web sémantique tels que le OWL et le *Semantic Web Rule Language* (SWRL).

L'ACO a été notamment utilisée dans les travaux de Héon (2010) pour la conception d'OntoCASE, un outil méthodologique et informatique de transformation de modèles de connaissances semi-formels en ontologies OWL. L'ACO a été aussi employée pour la conception de patrons de migration d'individu dans une ontologie (Héon et Rogozan, 2011). Un bref aperçu de ces recherches est présenté à la section 5.

Les figures de cet article sont exprimées dans le langage de *Modélisation par Objets Typés* (MOT) (Paquette, 2002). L'usage de MOT sera expliqué en bas de page au fur et à mesure de la présentation des modèles.

2. Architecture conduite par les modèles et les ontologies

Proposée par l'OMG, l'architecture conduite par les modèles (ACM) *"Model Driven Architecture*" offre un cadre méthodologique et architectural de développement de logiciels qui vise à découpler le développement des spécificités métiers des contraintes technologiques liées à l'implantation de la solution (Gašević,

[1] http://www.w3.org/TR/2004/REC-owl-guide-20040210/

Djurić et Devedžić, 2006 ; Kadima, 2005) Ce chapitre aborde le concept de couche d'abstraction, d'espace de modélisation ainsi que les concepts associés au processus de transformation de modèles selon l'ACM.

2.1. Couche d'abstraction et espace de modélisation

À l'instar de l'ACM, l'ACO se divise en quatre couches d'abstraction de M0 à M3 pour l'ACM et O0 à O3 pour l'ACO (voir la figure 1). La couche O0 sert à entreposer les données représentant la réalité à conceptualiser. Il s'agit des individus de l'ontologie. La couche de conceptualisation O1 (l'*ontologie*) définit les individus de la couche O0. La couche O2, la *métaontologie*, est une abstraction de la couche O1. La métaontologie définit le langage de représentation utilisé à la couche O1. Dans le projet du web sémantique, le OWL est un exemple de métaontologie. La couche O3, le *méta-métaontologie*, est la couche d'abstraction qui permet de construire les langages de modélisation de O2. La couche O3 est aussi réflexive, c.-à-d. qu'elle peut se définir elle-même ce qui permet, au besoin, d'accroître le nombre de couches d'abstraction. Dans le projet du web sémantique, le langage RDF/RDFS se situe à la couche O3.

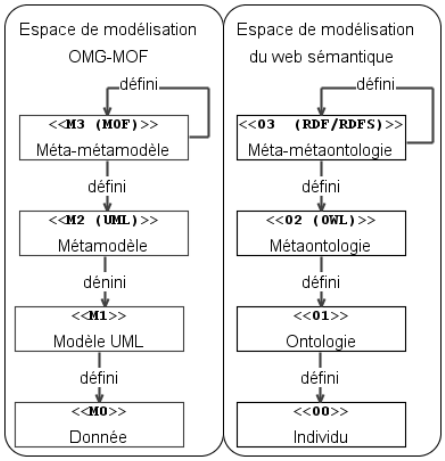


Figure 1. Couches d'abstraction de l'ACM et de l'ACO

L'*Espace de Modélisation* (EM) (Gašević, Djurić et Devedžić, 2006) est une architecture de modélisation réalisée à partir d'un méta-métamodèle donné. Présenté à la figure 1, l'EM-MOF (*Meta Object Facility*) offre l'architecture langagière pour la réalisation de modèles UML (*Unified Modeling Language*). De même, *l'espace de modélisation du web sémantique* (EM-WS) est l'architecture langagière de modélisation d'ontologies en OWL.

2.2. Processus de transformation de l'ACM

Une fonctionnalité importante de l'ACM est la transformation d'un modèle source vers un modèle cible. Présenté à la figure 2[2], le processus *Transformer* produit un *modèle cible* à partir de quatre intrants. Le premier intrant est le *modèle source* qui est à transformer en modèle cible. Pour réaliser la tâche de

[2] En langage MOT, l'ovale est utilisé pour représenter un *processus*, le rectangle pour représenter un *concept* et le lien IP pour représenter *l'intrant* ou le *produit* d'un processus. Le lien D est utilisé pour indiquer qu'un concept en *définit* un autre.

transformation, le processus utilise la structure du *langage de modélisation source* et la structure du *langage de modélisation cible*. Finalement, le processus de transformation s'exécute selon les déclarations représentées dans le *modèle de transformation*, qui lui-même est structuré selon un *langage de transformation*. Dans le EM-MOF le langage de transformation est définit par *Query / View / Transformation* combiné à *l'Object Constraint Language*. Le processus de transformation est réalisé par le *moteur de transformation* selon les directives représentées dans le modèle de transformation. Ainsi, le modèle de transformation est la représentation des règles métiers à exécuter pour la transformation.

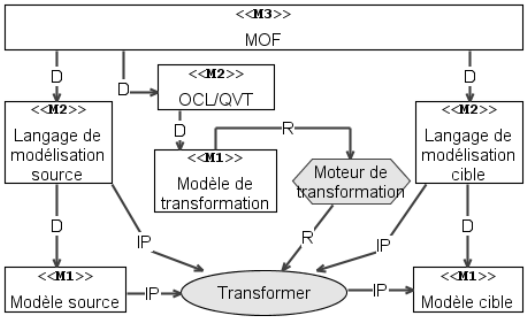


Figure 2. Processus de transformation dans l'ACM

2.3. ACM et le développement d'ontologies

Plusieurs recherches ont pour hypothèse d'utiliser l'ACM pour la construction d'ontologies. Notamment, dans son ouvrage, Gašević *et al.*(2006) utilisent le MOF pour définir l'*Ontology Definition Metamodel* (ODM) qui est la représentation UML du langage OWL. Pour chaque élément du modèle source, le processus de transformation attribue un stéréotype (définit dans l'ODM) à l'élément traduit dans le modèle cible. Par exemple, supposons la classe *Chien* dans le modèle source. Le processus de transformation traduira la classe *Chien* dans le modèle cible en lui attribuant le stéréotype « OWLClass ». À cette étape le modèle cible n'est pas encore une ontologie dans le formalisme OWL. Pour ce faire, une étape d'exportation doit être incluse dans le processus de construction de l'ontologie. L'exportation est une étape qui sert à exporter le modèle cible de l'EM-MOF vers une ontologie dans l'EM-WS. Ce n'est qu'après cette étape que l'ontologie produite sera utilisable dans l'environnement du web sémantique. On peut utiliser des implantations de l'ODM notamment chez IBM avec l'*Eclipse Ontology Definition Metamodel* (EODM) et chez la fondation *Eclipse* par le projet *ATL Usecase ODM Implementatioz.*

3. Question de recherche et hypothèse de travail

Dans cette recherche, nous soutenons qu'il est possible de transposer certains principes de l'ACM pour construire une architecture de transformation de modèles fondée sur les ontologies en utilisant les technologies associées au web sémantique. Nous émettons l'hypothèse, qu'il est possible de transformer des ontologies

directement dans l'EM-WS selon une *Architecture Conduite par les Ontologies ;*et que, la transformation est opérationnalisable par un moteur de transformation « intelligent » fondé sur une mécanique de raisonnement logique et à base de règles.

4. Moteur de transformation intelligent

Le moteur de transformation intelligent (MTI) est une application informatique développée selon le paradigme contemporain du développement logiciel notamment par la définition des exigences attendues, par l'explication du processus de transformation réalisé par le MTI, et par la présentation de l'architecture du MTI.

4.1. *Exigences*

Voici la liste des exigences attendues du moteur de transformation intelligent :

1- Le développement des règles de transformation, pour la programmation des règles métiers, doit être découplé de la programmation du MTI.
2- Le langage de programmation des règles doit être conforme aux langages du web sémantique notamment en OWL et SWRL.
3- Le MTI doit pouvoir traiter des documents dans le format OWL.
4- Le MTI doit pouvoir traiter des règles en groupes indépendants permettant de séquencer de manière prévisionnelle l'exécution des règles.
5- Le MTI doit pouvoir utiliser une version étendue des prédicats de SWRL.
6- Le MTI doit pouvoir créer une nouvelle ontologie avec tous ses composants (classes, propriétés, individus, axiomes, etc.).

4.2. Processus de transformation d'ontologies avec le MIT

Schématisé à la figure 3[3] le processus de transformation d'une ontologie réalisé par le MTI traduit une *ontologie source* (o-src) de couche O1 en une *ontologie cible* (o-cible) de couche O1. À la couche O3, le MOF est remplacé par le RDF(S)/OWL. La nature réflexive du OWL permet une combinaison avec le RDF(S) à la couche O3. La couche O2, celle du langage de domaine, contient : l'*Ontologie du langage du*

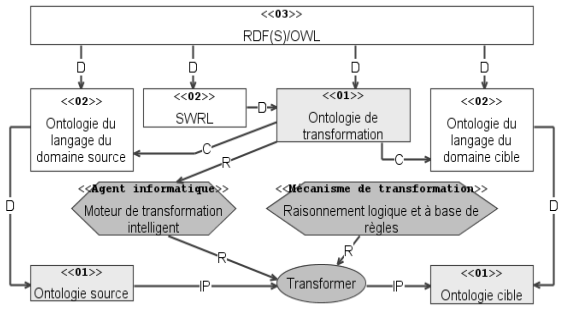


Figure 3. La transformation d'ontologies selon l'Architecture Conduite par les Ontologies

[3] En langage MOT, le principe (représenté par l'hexagone) est utilisé pour représenter une connaissance stratégique comme une condition, un agent ou encore une loi. Le lien R indique qu'un concept ou une procédure est *régi* par le principe.

domaine source nécessaire à la modélisation de l'o-src ; l'*Ontologie du langage du domaine cible* nécessaire à la modélisation de l'o-cible ; et le *SWRL* nécessaire à la modélisation des règles de transformation de l'*ontologie de transformation* (OT). Le déroulement du processus de transformation est guidé par l'OT, de couche O1, qui contient les axiomes et les règles pilotant l'exécution du MTI par des mécanismes de raisonnements logique et à base de règles. Insérée dans l'OT, l'ontologie du langage de domaine source et l'ontologie du langage de domaine cible permettent de construire des règles et des axiomes de transformation spécifiques aux domaines du discours représentés par l'o-src et l'o-cible.

4.3. Architecture du Moteur de Transformation Intelligent

Trois aspects composent l'architecture du MTI. Le premier aspect s'intéresse à la structure de l'OT qui contrôle l'activité de transformation. Le deuxième aspect, l'algorithme d'exécution de la transformation, présente les processus impliqués dans la transformation. Le troisième aspect, le mécanisme d'invocation des commandes *Java* à partir d'une inférence SWRL, présente l'architecture informatique supportant la transformation.

4.3.1. Ontologie de transformation

L'ontologie de transformation (OT), importée par l'o-src, est le volet déclaratif de l'opérationnalisation du MTI. C'est par l'OT que les opérations de transformation sont dictées au MTI. La figure 4[4] présente la structure interne de cette ontologie.

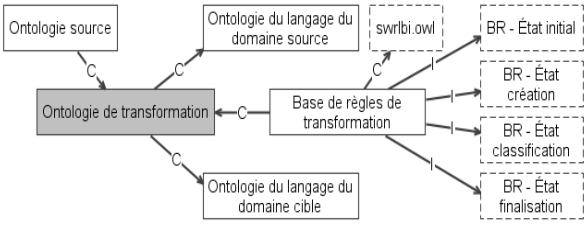


Figure 4. Structure de l'ontologie de transformation

L'OT qui importe l'*ontologie du langage du domaine source* et *l'ontologie du domaine cible,* structure la conception de règles de transformation selon la sémantique du domaine source et la sémantique du domaine cible. La *base de règles de transformation,* qui importe l'OT et l'ontologie *swrlbi.owl*[5], est une ontologie où sont déclarées les règles de transformation. L'ordre de déclenchement des règles est un phénomène imprévisible. Pour donner un certain ordre de séquencement dans l'exécution des règles, quatre instances de la base de règles de transformation

[4] En langage MOT, le lien I est utilisé pour lier une connaissance abstraite (comme un concept) à une connaissance factuelle (comme c'est ici le cas pour un énoncé [le rectangle en pointillé]). Le langage permet aussi de représenter une composition (lien C) et une composition multiple (lien C*).

[5] Cette ontologie contient la définition des commandes de type *SWRL-BuiltIn* qui sera présentée un peu plus loin dans cet article.

regroupent les règles en fonction de l'état d'exécution du MTI. La *BR – État initial* est la base de règles dont les objectifs sont d'assurer la création du fichier ontologique cible ainsi que l'initialisation de variables pouvant être utiles aux autres étapes de la transformation ; la *BR – État création* contient les règles nécessaires à la création des propriétés, classes, individus et axiomes de l'o-cible ; la *BR – État classification* contient les règles qui permettent de classifier les éléments de l'o-cible. C'est notamment à cette étape que les propriétés de type *subClassOf* ou *typeOf* sont attribuées aux éléments de l'o-cible. Finalement ;la *BR – État finalisation* regroupe l'ensemble de règles nécessaires à la conclusion du processus de transformation. Elle permet notamment de réaliser la sauvegarde de l'o-cible dans le fichier.

4.3.2. Algorithme d'exécution de la transformation intelligente

Présenté à la figure 5[6], l'algorithme d'exécution de la transformation intelligente débute par le processus de *lecture de l'o-src*. Pour chaque état (initial, création, classification et final) une *inférence logique* et une i*nférence à base de règles* sont appliquées à l'o-src. Guidée par l'*OT* et la *base de règles à la transformation*, l'exécution du mécanisme d'inférence produit les *résultats de l'inférence dans l'o-cible.* Un résultat peut être, par exemple, la création d'une nouvelle classe ou d'un nouvel individu, la classification d'une propriété ou toute autre action de manipulation d'éléments dans une ontologie.

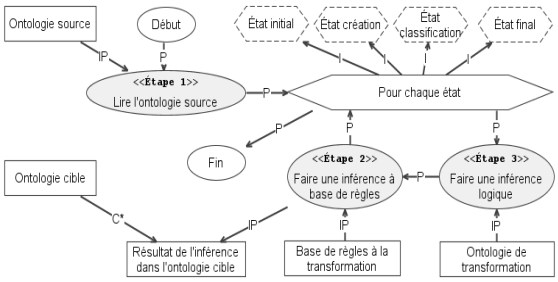


Figure 5. Algorithme d'exécution de la transformation intelligente

4.3.3. Mécanisme d'invocation des commandes Java à partir d'une inférence SWRL

Le MTI est une application *Java V1.5* qui utilise la bibliothèque de codes de *Protégé*[7] version 3.4.4. Cette bibliothèque fournit une interface au moteur *Pellet*[8] pour le raisonnement logique et au moteur *Jess*[9] pour le raisonnement à base de règles.

Le SWRL est un langage à base de règle structuré en *antécédent* qui implique un *conséquent.* L'antécédent et le conséquent sont des conjonctions d'*atomes* (par ex. :

[6] En MOT, on représente la préséance entre des procédures ou des conditions par un lien P.
[7] http://protege.stanford.edu/
[8] http://clarkparsia.com/pellet/
[9] http://www.jessrules.com/download.shtml

a_1^...^a_n) dont un type particulier est le *BuiltIn.* Selon les spécifications du SWRL (Horrocks *et al.*, 2004) l'atome *BuiltIn* est introduit afin d'offrir, à l'ingénieur ontologique, le moyen d'étendre les prédicats du langage. C'est cette stratégie qui est employée pour permettre la création d'une o-cible à partir du déclenchement de règles SWRL. L'édition des règles SWRL est réalisée grâce aux fonctionnalités du *SWRLTab*[10]. *Protégé* offre aussi une bibliothèque d'implantation *Java* dont le *SWRLBuitlinBridge* (O'Connor, 2009), qui permet d'associer une commande *SWRL-BuiltIn* à son implantation *Java.* Comme son nom l'indique, le *SWRLBuitlinBridge* est un mécanisme qui sert de pont entre l'atome SWRL et son implantation en *Java.* Ce mécanisme a été originellement conçu afin d'assurer le traitement des connaissances dans l'ontologie porteuse de la commande. Pour le MTI, nous élargissons cette propriété pour la création de nouvelles ontologies par l'implantation des patrons de conception *Command* et *Invoker* de Gamma *et al.* (1999).

Schématisé à la figure 6, les patrons *Command* et *Invoker* se composent de plusieurs constituants qui ont chacun un rôle spécifique. L'intention de ce patron est de fournir à un client un coffre à outil d'objets qui sont en fait des actions traitables. L'interface *Command* est un objet abstrait qui permet d'encapsuler l'exécution d'une commande dont l'implantation est réalisée par la classe *ConcreteCommand* (voir l'exemple du tableau 1 **d**). L'exécution de l'action de la *ConcreteCommand* est réalisée par l'appel du *Receiver* qui implémente l'action à accomplir. Le *Client* est l'application qui instancie la *ConcreteCommand* pour ensuite la transmettre à l'*Invoker*, responsable du déclenchement de l'exécution de la *Command*.

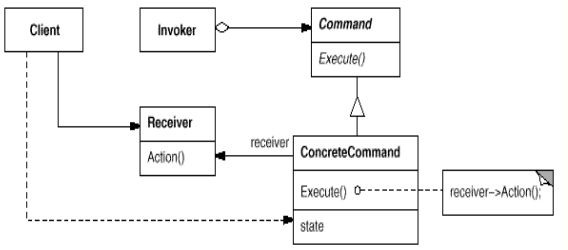


Figure 6 : Diagramme UML des composants du patron de conception *command* et *invoker* (Tiré de Gamma *et al.*, 1999)

Les patrons sont couplés avec le dispositif d'appel de commandes SWRL-*BuiltIn* qui est interfacé grâce au *SWRLBuitlinBridge* de *Protégé*. Le tableau 1 présente un aperçu de la structure interne des divers modules du MTI. Au tableau 1 **a**, est présenté la signature de l'atome *BuiltIn* de l'invocateur. Le « *swrlbi*: » fait référence au nom de domaine de l'ontologie « swrlbi.owl » à l'intérieur duquel est définit l'atome *BuiltIn* alors que l'« *invoker* » est le nom de l'atome permettant d'appeler l'invocateur. En tant que paramètre à l'invocateur, le *nomDeLaCommande* fait référence à l'une des commandes de manipulation de l'o-cible développée en *Java* (la section **e** du tableau 1 présente une liste de commandes pouvant être invoquées).

[10] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab

L'*argument* est un paramètre passé à la commande. En **b,** est présenté un exemple d'invocation de la commande *OWLCreerUneClasseCmd.* En **c,** est présenté la déclaration *xml* de L'« *invoker* » dans l'ontologie « `swrlbi.owl` » alors que **d,** présente le contenu du programme *Java* qui réalise la commande.

a) Signature SWRL et exemple d'invocation d'une commande

```
swrlbi:invoker(nomDeLaCommande,argument1, ..., argumentn)
```

b) Appel de la commande Java **OWLCreerUneClasseCmd** à partir d'une règle SWRL de création d'une classe dans une ontologie

```
swrlbi:invoker("OWLCreerUneClasseCmd", ?nc)
```

c) Déclaration de l'atome *invoke* dans le fichier **`swrlbi.owl`**

```
<swrl:Builtin rdf:ID="invoker">
  <swrlb:minArgs rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</swrlb:minArgs>
</swrl:Builtin>
```

d) Implantation de l'action « créer une classe OWL » de la commande Java **`OWLCreerUneClasseCmd`**

```
public Boolean action() {
    owlModel = ontologieRegister.getOwlDomainModel();
    OWLNamedClass sousClasse =
        owlModel.createOWLNamedClass(nomClasse);
    return new Boolean(true);  }
```

e) Liste de quelques commandes de la boite à outils pour la construction d'une o-cible

OWLAjoutDuneProprieteEntreIndividuCmd.java	OWLCreerUneClasseCmd.java
OWLAssignerRestrictionAPourValeur_FloatCmd.java	OWLCreerUneClasseSousCmd.java
OWLAssignerRestrictionAPourValeur_IndividuCmd.java	OWLCreerUneInstanceSousCmd.java
OWLAssignerRestrictionAPourValeur_IntegerCmd.java	OWLCreerUneProprieteCmd.java
OWLAssignerRestrictionAPourValeur_StringCmd.java	OWLEstDisjointDeCmd.java
OWLAssignerRestrictionExistentielleCmd.java	OWLEstUneInstanceDeCmd.java
OWLAssignerRestrictionUniverselleCmd.java	OWLEstUneSousClasseDeCmd.java
OWLProprieteSetFonctionelleCmd.java	OWLEstUneSousProprieteDeCmd.java
OWLPropriete_AjoutDunDomaineCmd.java	OWLSupprimerSuperClasseDeCmd.java
OWLPropriete_AjoutDunDomaineEtDuneImageCmd.java	printf.java
OWLSupprimerLeTypeDeLinstanceCmd.java	SauvegarderOntologieCmd.java
OWLSupprimerSuperProprieteDeCmd.java	OWLProprieteSetInverseOfCmd.java
OWLCreerUneProprieteDatatypeCmd.java	OWLProprieteSetSymetricCmd.java
OWLProprieteSetInvFonctionelleCmd.java	OWLProprieteSetTransitiveCmd.java

Tableau 1. Aperçu des éléments du mécanisme d'invocation du MTI

5. Expérimentation

Deux projets ont permis de valider l'ACO et le MTI. Le premier projet a pour objet la thèse OntoCASE (Héon, 2010) dont le sujet porte sur la formalisation de modèles semi-formel en ontologies formelles. Le deuxième projet vise le développement de règles de migration des instances d'une o-src vers une o-cible (Héon et Rogozan, 2011).

5.1. Formalisation d'un modèle semi formel en ontologie

OntoCASE est une méthodologie accompagné par un agent informatique intelligent qui vise la construction d'une ontologie formelle à partir de la formalisation d'un modèle semi-formel. Le modèle de la figure 7 présente une vue d'ensemble de la méthodologie. Dans l'EM-MOF, un certain modèle semi-formel en langage MOT est produit. Le processus d'*importation* du modèle MOT traduit le modèle de l'EM-MOF à l'EM-WS. Pour formaliser le modèle semi-formel, le processus de *formalisation* réalise une étape de *désambiguïsation*, puis, une étape de

conversion. La formalisation est contrôlée par le MTI selon les connaissances représentées dans l'OT d'OntoCASE.

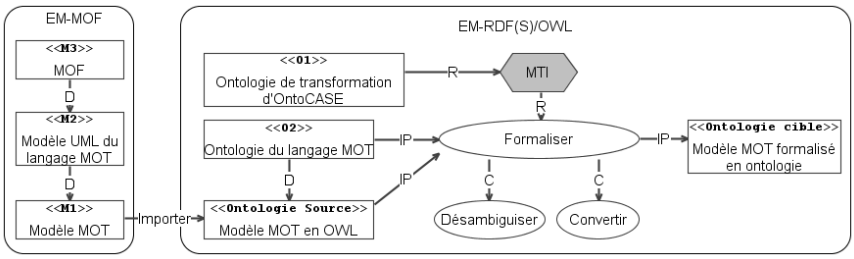


Figure 7: ACO et le MTI dans la problématique traitée par OntoCASE

Le cas présenté au tableau 2 est un exemple de transformation d'une o-src en o-cible dans un contexte de formalisation d'un modèle semi-formel en ontologie.

a) Exemple d'un modèle d'origine en MOT dans l'EM-MOF

« Le processus2 P a pour intrant A et a pour produit B »

b) Ontologie source en OWL N-3

```
:A_0  rdf:type metaMot:MOT_Concept ;
  metaMot:MOT_estLaSource :LienIP_A_0_P_1 ;
  metaMot:MOT_etiquette "A"^^xsd:string ;
  metaMot:MOT_nomConnaissance
      "A_0"^^xsd:string .
:B_2  rdf:type metaMot:MOT_Concept ;
  metaMot:MOT_estLaDestination
      :LienIP_P_1_B_2 ;
  metaMot:MOT_etiquette "B"^^xsd:string ;
  metaMot:MOT_nomConnaissance
      "B_2"^^xsd:string .
:LienIP_A_0_P_1
  rdf:type metaMot:MOT_LienIP ;
  metaMot:MOT_connDestination :P_1 ;
  metaMot:MOT_connSource :A_0 ;
  metaMot:MOT_nomLien
      "LienIP_A_0_P_1"^^xsd:string .
:LienIP_P_1_B_2
  rdf:type metaMot:MOT_LienIP ;
  metaMot:MOT_connDestination :B_2 ;
  metaMot:MOT_connSource :P_1 ;
  metaMot:MOT_nomLien
      "LienIP_P_1_B_2"^^xsd:string .
:P_1  rdf:type metaMot:MOT_Procedure ;
  metaMot:MOT_estLaDestination
      :LienIP_A_0_P_1 ;
  metaMot:MOT_estLaSource :LienIP_P_1_B_2 ;
  metaMot:MOT_etiquette "P"^^xsd:string ;
  metaMot:MOT_nomConnaissance
      "P_1"^^xsd:string .
```

c) Ontologie cible en OWL N-3

```
:A_0  rdf:type owl:Class ;
      rdfs:label "A"^^xsd:string ;
      rdfs:subClassOf owl:Thing ,
          metaDom:MD_Declarative_Concept .
:B_2  rdf:type owl:Class ;
      rdfs:label "B"^^xsd:string ;
      rdfs:subClassOf owl:Thing ,
          metaDom:MD_Declarative_Concept .
:P_1  rdf:type owl:Class ;
      rdfs:label "P"^^xsd:string ;
      rdfs:subClassOf owl:Thing ,
          metaDom:MD_Procedurale_Procedure .
:P_1_aPourIntrant_A_0
      rdf:type owl:ObjectProperty ;
      rdfs:domain :P_1 ;
      rdfs:label ""^^xsd:string ;
      rdfs:range :A_0 ;
      rdfs:subPropertyOf
          metaDom:A-POUR-INTRANT .
:P_1_aPourProduit_B_2
      rdf:type owl:ObjectProperty ;
      rdfs:domain :P_1 ;
      rdfs:label ""^^xsd:string ;
      rdfs:range :B_2 ;
      rdfs:subPropertyOf
          metaDom:A-POUR-PRODUIT .
```

Tableau 2. Exemple de transformation d'un modèle source en o-cible

Le modèle d'origine, en **a**, contient cinq éléments de modèles à formaliser soit : deux concepts (A et B), une procédure (P) et deux liens intrant/produit (IP). La polysémie (plusieurs significations) que l'on attribue à certains types d'élément de modèle est la caractéristique qui distingue le modèle semi-formel d'un modèle

formel. Ici, le lien IP porte une double signification. Entre A et P, il désigne un *intrant* alors qu'entre P et B il désigne un *produit*. Chacun des éléments du modèle sont importés dans l'EM-WS et sont représentés dans le formalisme OWL (voir le tableau 2 **b**). La première étape de la formalisation consiste à désambiguïser les éléments du modèle. Par exemple, le MTI désignera que le lien IP entre A et P est un lien de type *Intrant.* À l'étape de conversion, le MTI produira l'o-cible selon le type de chacun des éléments du modèle. Par exemple (voir le tableau 2 **c**) : A et B seront convertis en *owl:class* de la catégorie *metaDom:MD_Declarative_Concept,* P sera aussi converti en *owl:class* dans la catégorie *metaDom: MD_Procedurale_Procedure* et les liens IP sont convertis en *owl:ObjectProperty* de type *A-POUR-INTRANT* ou *A-POUR-PRODUIT* selon le cas.

Comme nous l'avons précédemment vu, l'exécution du MTI passe par quatre états comportant chacun un ensemble de règles spécifiques, Le tableau 3 présente l'exemple d'une règle servant à la conversion en o-cible pour chaque état du MTI.

État d'exécution	**Exemples d'une règle de conversion déclenchée pour chaque état d'exécution**
État initial	`ot:OT_DataOntologieDeDomaine(?o) ∧ ot:OT_nameSpace(?o, ?nd) -> swrlbi:invoker("CreerUneOntologieCmd", ?nd)`
	SOIT une ontologie *?o* , obtenir le nom du domaine *?nd* ALORS, créer une ontologie dont le nom de domaine est *?nd*
État création	`oRef:OR_Entite_Concept(?c) ∧ oRef:OR_identifiant(?c, ?nc) ∧ oRef:OR_etiquette(?c, ?e) -> swrlbi:invoker("OWLCreerUneClasseCmd", ?nc, ?e)`
	SOIT une entité *?c* de type « Concept », obtenir son identifiant *?nc*, obtenir son étiquette *?e* ALORS Créer une classe du nom de ?*nc* avec une étiquette *?e*
État classification	`oRef:OR_Relation_Flux_Produit(?lip) ∧ oRef:OR_connSource(?lip, ?src) ∧ oRef:OR_connDestination(?lip, ?dest)∧ oRef:OR_Entite_Concept_Classe(?dest) ∧ oRef:OR_Entite_Action(?src) ∧ oRef:OR_identifiant(?src, ?nomSrc) ∧ oRef:OR_identifiant(?dest, ?nomDest) ∧ swrlb:stringConcat(?nomPropCompo, ?nomSrc, "_aPourProduit_", ?nomDest)-> swrlbi:invoker("OWLPropriete_AjoutDunDomaineEtDuneImageCmd", ?nomSrc, ?nomPropCompo, ?nomDest)`
	SOIT un lien IP *?lip* qui est un produit, et qui unit une connaissance source *?src* de type classe à une connaissance destination *?dest* de type action ET qui ont respectivement les identifiants *?nomSrc* et *?nomDest* ET une variable *?nomPropCompo* qui contient le nom de la propriété à traiter ALORS, ajouter à la *?nomPropCompo* le domaine *?nomSrc* et l'image *?nomDest*
État final	`ot:OT_DataOntologieDeDomaine(?o) ∧ ot:OT_nameSpace(?o, ?ns) ∧ ot:OT_defaultUriLocation(?o, ?nf) -> swrlbi:invoker("SauvegarderOntologieCmd", ?nf, ?ns) ∧ swrlbi:invoker("printf", "Sauvegarde du fichier (%s)", ?nf)`
	SOIT l'ontologie de domaine *?o* et son « nameSpace » *?ns* et son URI *?nf* ALORS sauvegarder l'ontologie et imprimer le message « *Sauvegarde du fichier ?nf* »

Tableau 3. Exemples de règles de transformation pour chacun des groupes de base de règles

Cette recherche a permis de construire dans l'EM-WS un outil qui automatise et semi-automatise le processus de formalisation d'un modèle semi-formel tout en représentant le processus de formalisation dans une ontologie de transformation.

5.2. Migration d'individus d'ontologie

La migration d'individus d'ontologie (Héon et Rogozan, 2011) est un projet dont la visée est d'automatiser la préservation et la migration des individus d'une o-src possédant une certaine structure vers une o-cible dont la structure a évoluée. L'ACO et le MTI ont été mis à profit afin d'identifier et valider des patrons de migration servant à généraliser le processus de migration. Dans cet article, nous limitons la discussion sur l'apport de l'ACO et du MTI dans le projet. L'ACO, schématisée à la figure 8, présente les composants nécessaires à la réalisation de la migration des individus de l'o-src vers l'o-cible. Le MTI contrôle le processus de migration selon les spécificités de l'ontologie de migration et de la base de règles de migration.

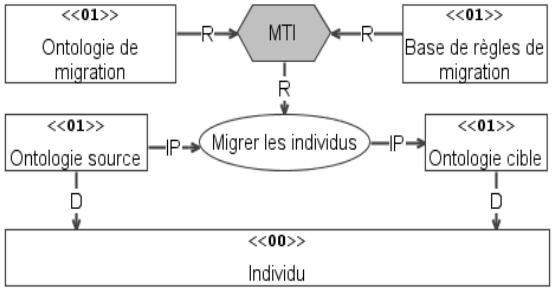


Figure 8 : Architecture ontologique du processus de migration des individus

La première phase d'exécution prévoie le déclenchement de la base de règles responsables de la migration des individus par la réaffectation de la typologie des individus alors que la deuxième phase s'adresse au déclenchement de la base de règles responsable de la migration des prédicats qui uni les individus.

Dans l'exemple du patron de migration, *fusionner les propriétés,* présenté au Tableau 4, l'o-src contient deux propriétés[11] (P1 et P2) qui sont fusionnées en une seule propriété (Q) dans l'o-cible. La première phase d'exécution du MTI est de migrer les individus. Par exemple : l'individu a1, de la classe A1 de l'o-src (`src:A1(a1)`) vers A1 de l'o-cible (`cible:A1(a1)`), ainsi de suite pour a2, b1 et b2. La deuxième phase d'exécution de la migration consiste à remplacer l'assignation des prédicats (P1 et P2) de l'o-src au prédicat (Q) de l'o-cible.

[11] Le langage MOT est utilisé pour schématiser la structure d'une ontologie. Le rectangle s'utilise pour représenter l'*owl :Class* et l'hexagone s'utilise pour représenter une *owl:objectProperty.* Le lien R est utilisé pour indiquer le domaine et l'image de la propriété.

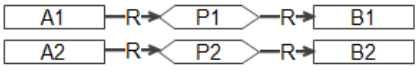
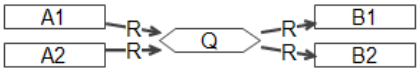
Structure de l'ontologie source			Structure de l'ontologie cible		
					
sujet	prédicat	objet	sujet	prédicat	objet
src:A1(a1)	P1	src:B1(b1)	cible:A1(a1)	Q	cible:B1(b1)
src:A2(a2)	P2	src:B2(b2)	cible:A2(a2)	Q	cible:B1(b2)

Tableau 4 : Affectation des individus et structure de l'ontologie source et de l'ontologie cible pour le patron de migration *fusionner les propriétés*

Le tableau 5 présente la règle qui permet de migrer le prédicat de chacun des individus. Le préfixe `omigr:` indique que l'atome est définit dans l'ontologie de migration le rendant ainsi réutilisable pour d'autres patron de migration.

Antécédent de la règle:

```
1.  omigr:isOneOfEachString(?nomProp, "P1", "P2") ∧
2.  omigr:isSrcPropertyOfName(?srcProp, ?nomProp) ∧
3.  omigr:isTargetPropertyOfName(?cibleProp, "Q") ∧
4.  omigr:isInDirectDomainOf(?domaine, ?srcProp) ∧
5.  omigr:isInDirectRangeOf(?image, ?srcProp) ∧
6.  omigr:isInstanceOf(?sujet, ?domaine) ∧
7.  omigr:isInstanceOf(?objet, ?image) ∧
8.  omigr:hasPredicate(?sujet, ?srcProp, ?objet) ∧
9.  omigr:hasCorrespondingIndividual(?sujet, ?sujet_cible) ∧
10. omigr:hasCorrespondingIndividual(?objet, ?objet_cible)
```

Conséquent de la règle:

```
11. omigr:hasPredicate(?sujet cible, ?cibleProp, ?objet cible)
```

Tableau 5 : Règle de migration des prédicats de la base de règle de migration pour le patron *fusionner les propriétés*

Ce projet a permis de développer près d'une dizaine de patrons de migration (par ex.: *Diviser une classe selon la valeur d'une propriété, Fusionner une classe par l'image d'une propriété*, etc.) et une dizaine de prédicats de migration (par ex. : *isInstanceOf, hasPredicate, isSrcIndividual* et autres) qui servent d'atomes à la conception des règles de migration.

6. Conclusion

Cet article a présenté l'implantation d'un moteur de transformation intelligent qui supporte la transformation d'ontologies selon le principe d'architecture conduite par les ontologies. l'ACO et le MTI ont été utilisé dans deux projets de recherche distinctes. Du point de vue des exigences exprimées à la section 4.1, l'utilisation des patrons de conception *Command* et *Invoker* permet de découpler le mécanisme de programmation des règles de celle de la programmation d'implantation du MTI. De plus, ces patrons de conceptions permettent de construire un système automatique de construction d'ontologies. Effectivement, au-delà d'une simple transformation d'ontologie, OntoCASE permet d'automatiser la *création* d'une o-cible. Le

mécanisme d'exécution par *état* du MTI permet le regroupement de règles et l'exécution de celles-ci de manière spécifique et ordonnée. Cependant, cette stratégie d'exécution complexifie le processus de déverminage des règles à cause de la forte interdépendance des différentes bases de règles. Par exemple, dans la phase de désambiguïsation, OntoCASE utilise près de 600 règles réparties dans ses quatre bases de règles. Pour déverminer un tel système, des cas de tests formels doivent donc être développés pour assurer la cohérence des règles entre les différents états complexifiant ainsi la programmation des règles.

Au moment de l'écriture de cet article, une restriction certaine quant au nombre de moteurs de raisonnement SWRL disponible est observée. Outre *Protégé*, Pellet traite aussi les atomes *BuiltIn*. Mais là encore les possibilités sont restreintes aux *BuiltIn* développés pour ce moteur spécifique et aucune interface de développement ne semble disponible pour étendre les fonctionnalités *BuiltIn*. Une autre limitation observée est le manque d'éditeur de règles SWRL. Au-delà de *Protégé,* il y a *TopBraid Composer* qui supporte l'édition de commande SWRL standard sans supporter l'édition d'atome *Builtin*. Finalement, nos essais ont été réalisés avec des ontologies comportant quelques dizaines de classes, de propriétés et d'individus. Des tests plus exhaustifs sur des ontologies contenant des milliers d'éléments devraient être pratiqués afin de valider le passage à des échelles de traitement plus grand.

Bibliographie

Gamma, Erich, Richard Helm, Ralph Johnson et John Vlissides. 1999. *Catalogue de modèles de conception réutilisables*. Jean-Marie Lasvergères: Vuibert.

Gašević, Dragan, Dragan Djurić et Vladan Devedžić. 2006. *Model Driven Architecture and Ontology Development*. New York, Inc.: Springer-Verlag.

Héon, Michel. 2010. «OntoCASE: Méthodologie et assistant logiciel pour une ingénierie ontologique fondée sur la transformation d'un modèle semi-formel». Thèse, Montréal, Thèse en Informatique Cognitive, Université du Québec à Montréal. En ligne. <http://www.cotechnoe.com/>.

Héon, Michel, et Delia Codruta Rogozan (2011). Patrons de migration des individus d'une ontologie. Rapport de recherche. Montréal, Centre de recherche LICEF de la TELUQ

Horrocks, Ian, Harold Boley, Said Tabet, Benjamin Grosof et Mike Dean. 2004. «SWRL: A Semantic Web Rule Language Combining OWL and RuleML». W3C. En ligne. <http://www.w3.org/Submission/SWRL/>. Consulté le 29 mai.

Kadima, Hubert. 2005. *MDA: Conception orientée objet guidée par les modèles*. Coll. «Études & Développement». Paris: Dumond.

Miksa, Krzysztof, Pawel Sabina et Marek Kasztelnik. 2010. «Combining Ontologies with Domain Specific Languages: A Case Study from Network Configuration Software». Dans *Reasoning Web. Semantic Technologies for Software Engineering,* Uwe Aßmann, Andreas Bartho et Christian Wende, p. 99-118: Springer Berlin / Heidelberg.

O'Connor, Martin. 2009. «SWRLBuilt In Bridge». Stanford Center for Biomedical Informatics Research. En ligne. <http://protege.cim3.net/cgi-bin/wiki.pl?SWRLBuiltInBridge>. Consulté le 09/01/09.

Paquette, Gilbert. 2002. *Modélisation des connaissances et des compétences : un langage graphique pour concevoir et apprendre*. Sainte-Foy: Presses de l'UQ.

Staab, Steffen, Tobias Walter, Gerd Gröner et Fernando Parreiras. 2010. «Model Driven Engineering with Ontology Technologies». Dans *Reasoning Web. Semantic Technologies for Software Engineering,* Uwe Aßmann, Andreas Bartho et Christian Wende, p. 62-98: Springer Berlin / Heidelberg.