HAL
open science

Gestion de la cohérence des règles métier éditées à partir d'ontologies OWL

Amina Chniti, Patrick Albert, Jean Charlet

To cite this version:

Amina Chniti, Patrick Albert, Jean Charlet. Gestion de la cohérence des règles métier éditées à partir d'ontologies OWL. IC 2011, 22èmes Journées francophones d'Ingénierie des Connaissances, May 2012, Chambéry, France. pp.589-606. hal-00746726

HAL Id: hal-00746726

https://hal.science/hal-00746726v1

Submitted on 29 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Gestion de la cohérence des règles métier éditées à partir d'ontologies OWL ⋆

Amina Chniti[1,2], Patrick Albert[1], Jean Charlet[2,3]

[1] IBM, CAS France
{amina.chniti,albertpa}@fr.ibm.com

[2] INSERM UMRS 872, Eq 20, 15, Rue de l'école de médecine, 75006, Paris, France
jean.charlet@upmc.fr

[3] AP-HP, Paris, France

Résumé : Les ontologies permettent la modélisation des connaissances d'un domaine. Elles sont généralement écrites dans un langage technique, assimilé uniquement par un expert, et sont amenées à évoluer au cours du temps. Les règles métier permettent de décrire les actions à mener pour un processus métier donné et sont généralement écrites dans un langage naturel contrôlé. Cet article décrit une approche qui permet de gérer la cohérence des règles métier éditées à partir d'ontologies OWL lors de leurs évolutions. Deux méthodes sont présentées, la première permet l'édition des règles métier, en langage naturel contrôlé, à partir d'ontologies OWL et la deuxième permet de gérer la cohérence de ces règles lors de l'évolution de l'ontologie correspondante.

Mots-clés : Ontologie, Règle Métier, cohérence.

1. Introduction

Les ontologies sont de plus en plus utilisées pour la modélisation des connaissances d'un domaine et sont souvent modélisées dans des langages techniques qui ne peuvent être assimilés par les utilisateurs métier.

Les règles métier [1] permettent de modéliser une décision métier. Elles capitalisent les connaissances d'une entreprise et traduisent sa stratégie en dé-

⋆. Ce travail est partiellement financé par le projet européen, ONTORULE (IST-2009-231875)

1. http ://www.businessrulesgroup.org/

crivant les actions à mener pour un processus donné. Elles sont généralement écrites dans un langage naturel contrôlé.

Le travail effectué propose une méthode pour la gestion des relations entre les ontologies et les règles métier en particulier dans le cadre du processus d'édition et de gestion de la cohérence des règles métier. L'objectif de notre travail est de permettre à des utilisateurs métier de :

- modéliser les connaissances de leur domaine dont la sémantique est formalisée en OWL, au moyen de règles éditées en langage naturel contrôlé ;
- gérer l'impact de l'évolution de l'ontologie modélisant leur domaine tout en garantissant l'évolution et la gestion de la cohérence de l'ensemble des règles éditées à partir de cette ontologie.

La problématique principale traitée dans cet article est la gestion de la cohérence des règles métier éditées à partir d'une ontologie OWL lors de l'évolution de cette dernière. Les règles dépendent des concepts et des propriétés de l'ontologie, de ce fait son évolution a un impact sur l'ensemble des règles et peut générer des problèmes d'incohérence. Pour cela, nous avons commencé par éditer des règles métier à partir d'ontologies OWL ensuite nous avons simulé des changements sur ces ontologies et analyser l'impact de ces changements sur les règles pour ensuite détecter les incohérences qui peuvent être générées.

L'article est organisé comme suit, la section 2. décrit le prototype développé pour l'édition des règles métier à partir d'une ontologie OWL. La section 3. présente la méthode utilisée pour la gestion de la cohérence des règles métier lors de l'évolution de l'ontologie. La section 4. décrit les résultats obtenus pour la gestion de la cohérence des règles métier. La section 5. discute notre approche et la compare aux travaux existants. Finalement la section 6. décrit nos perspectives et conclut.

2. Édition des règles métier à partir d'ontologies OWL

Les ontologies fournissent un socle théorique et pratique pour une modélisation robuste d'un domaine alors que les règles métiers fournissent les formalismes et les outils pour utiliser ces modèles afin d'automatiser des décisions. Cette section décrit la méthode développée pour éditer des règles métier, en langage naturel contrôlé, à partir d'ontologies, ce qui permet aux utilisateurs métier de modéliser des décisions sur des domaines dont la sémantique est formalisée en OWL. Une ontologie décrit les concepts et les propriétés d'un domaine. Une règle métier utilise les concepts et les propriétés de l'ontolo-

gie pour décrire les actions à mener pour un processus donné. Par exemple, à partir d'une ontologie qui modélise un système de validation de prescription pharmaceutique, nous voulons modéliser la décision de validation d'une prescription en fonction du nom du médicament prescrit et de la posologie prescrite. Cette ontologie (voir figure 2), contient les concepts `Prescription` pour un `Médicament` qui a un `nom de spécialité`. Une prescription a une `posologie` et est validée par un `MedecinPrescripteur`. La règle métier qui permet de modéliser le processus de validation d'une prescription correspondant à cette description sera comme suit

```
r1: si le nom de spécialité du médicament de la prescription
    est "Glucophage" et la posologie de la prescription est 4
    alors assigner l'avis du medecin prescripteur
    de la prescription à "Prescription invalide";
```

Pour permettre l'édition des règles métier, à partir d'ontologies, nous avons utilisé le `Système de Gestion de Règles Métier` (SGRM) IBM WebSphere ILOG JRules (voir section 2.1.) et effectué une transformation du modèle OWL vers le formalisme de modélisation de domaine de JRules (voir section 2.2.).

2.1. IBM WebSphere ILOG JRules

JRules [2] est un système de gestion de règle métier qui permet à des utilisateurs métier d'éditer, exécuter et mettre à jour, de manière collaborative, des règles métier écrites en langage naturel contrôlé. Le processus d'édition des règles métier dans JRules, commence d'abord par la génération d'un `Business Object Model` (BOM). Le BOM est un modèle orienté-objet, proche des diagrammes de classe UML. Il décrit les concepts et les propriétés d'un domaine et est représenté sous forme d'un ensemble de classes, correspondant aux concepts du domaine, et chaque classe contient un ensemble d'attributs, correspondant aux propriétés des concepts. Les composants du BOM (classes et attributs) définissent les entités sur lesquelles les règles vont être éditées. Deux modèles sont nécessaires pour l'édition des règles. Le premier étant le BOM qui modélise les entités du métier (e.g. `Prescription`, `posologie`) et le deuxième est le `vocabulaire` (VOC). Le VOC ajoute

2. http ://publib.boulder.ibm.com/infocenter/brjrules/v7r1/index.jsp ?topic=/com.ibm.websphere.ilog.jrules.doc/Content/Business-Rules/Documentation/-pubskel/JRules/ps-JRules-Global.html

un niveau de terminologie au dessus du BOM, ce qui permet d'éditer le texte des règles dans un langage naturel contrôlé (e.g. « `la prescription` » , « `la posologie de la prescription` »).

2.2. Méthode utilisée

JRules offre une infrastructure pour éditer des règles métier, en langage naturel contrôlé. L'idée est d'exploiter cette infrastructure et de l'adapter aux ontologies OWL. Cette adaptation consiste à effectuer une transformation automatique du modèle OWL vers le BOM de JRules. Ceci permet d'importer des ontologies écrites en OWL et de générer le BOM correspondant à chaque ontologie. Une fois le BOM généré, toutes les fonctionnalités de JRules, telles que l'édition et l'exécution des règles, peuvent être utilisées sans aucun changement (voir figure 1).

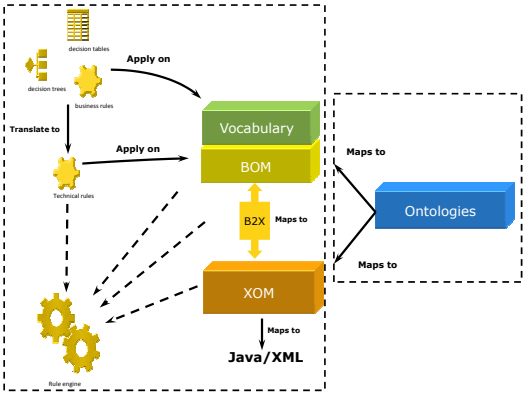


FIGURE 1: Architecture générale

Il existe des travaux de transformation de OWL vers Java tels que Kazuki [3], OntoJava [4], Owl2Java [5], ainsi que la méthode décrite dans (Kalyanpur *et al.*, 2004a). Pour effectuer la transformation du modèle OWL vers le BOM nous nous sommes basés sur la méthode décrite par (Kalyanpur *et al.*, 2004a) que nous avons adapté à nos besoins (Chniti *et al.*, 2010).

3. http ://projects.semwebcentral.org/projects/kazuki/
4. http ://www.aifb.uni-karlsruhe.de/WBS/aeb/ontojava/
5. http ://www.incunabulum.de/projects/it/owl2java.

3. Gestion de la cohérence des règles métier

Une ontologie évolue tout au long de son cycle de vie. Cette évolution consiste en un ensemble de changements tels que la restructuration de la conceptualisation, la dynamicité du domaine modélisé, les changements des besoins utilisateurs ... Les règles métier dépendent des concepts et des propriétés de l'ontologie. De ce fait, l'évolution d'une ontologie a un impact sur la cohérence de l'ensemble des règles. Dans ce qui suit, nous commencerons par introduire la notion de cohérence d'un ensemble de règles (section 3.1.), ensuite nous décrirons la méthode utilisée pour la gestion de la cohérence lors de l'évolution de l'ontologie correspondante (section 3.2.).

3.1. Cohérence d'un ensemble de règles

Un ensemble de règles est dit cohérent s'il ne contient aucune incohérence. Dans notre travail, nous distinguons cinq types d'incohérences qui peuvent impacter un ensemble de règles tels que la contradiction, les règles redondantes, les règles équivalentes, la violation de domaine et les règles jamais applicables (Pührer *et al.*, 2010). Dans ce papier, nous discutons de deux types : la violation de domaine et les règles jamais applicables.

– Violation de domaine : une incohérence de type violation de domaine est détectée dans un ensemble de règles, si cet ensemble contient au moins une règle qui assigne à un attribut, une valeur hors de son domaine.
  Exemple : soit la propriété `unitéTemps` définie avec la restriction `allValuesFrom{"TLJ","A", "B", "C"}` à laquelle nous appliquons un changement qui supprime la valeur « TLJ » du domaine de la restriction et la règle $r2$ ci-dessous :

  ```
  r2 : si la posologie du schema posologique de la pres-
       cription du patient est plus grande que 3 et le
       DFG du résultat biologique du patient est
       plus grand que 80
       alors assigner l'unité de temps du schema posolo-
       gique de la prescription du patient à "TLJ";
  ```

  La règle $r2$ assigne à la variable `unitéTemps` la valeur "TLJ" (Tous Les Jours) qui, une fois le changement appliqué, sera hors du domaine de la restriction de la propriété ce qui cause la violation de domaine.

– Règles jamais applicables : une règle n'est jamais applicable si elle ne

peut jamais être exécutée. Une règle n'est jamais exécutée si sa prémisse n'est jamais vérifiée.
Exemple : soit la définition de la propriété `unitéTemps` et le changement ci-dessus et la règle $r3$ ci-dessous :

```
r3 : si le nom de spécialité du médicament est "GLUCOPHAGE
     1000MG CPR" et l'unité de temps du schema posologique
     de la prescription du patient est "TLJ"
     alors assigner l'unité dosage du schema posologique
     de la prescription du patient à "COMPRIME(S)";
```

Cette règle ne sera jamais applicable car la propriété `unitéTemps` ne peut avoir la valeur "TLJ". De ce fait, la règle $r3$ ne sera jamais exécutée.

3.2. Approche de la gestion de la cohérence

L'approche adoptée pour la gestion de la cohérence d'un ensemble de règles lors de l'évolution de l'ontologie correspondante est inspirée de ONTO-EVO^{A}L (Djedidi & Aufaure, 2010) qui permet la gestion de la cohérence d'une ontologie lors de son évolution. Nous utilisons donc une approche à base de patrons appelés **Patrons de Gestion de Changement** qui consistent en trois catégories de patrons :

- **Patrons de changement** : permettent de modéliser des changements ;
- **Patrons d'incohérence** : permettent de détecter les incohérences causées par un changement ;
- **Patrons de réparation** : permettent de réparer les incohérences.

3.2.1. Processus de gestion de la cohérence

L'objectif du processus de gestion de la cohérence est d'assurer l'évolution et la cohérence d'un ensemble de règles, de manière automatique, lors de l'évolution l'ontologie correspondante. Ceci nécessite de formuler le changement, détecter les incohérences qui peuvent éventuellement être causées et proposer des réparations pour les résoudre (Stojanovic, 2004). Le processus est donc constitué de trois étapes :

1. **Spécification du changement**
   La phase de spécification de changement est déclenchée lors de la détection d'un changement sur l'ontologie. Cette phase consiste à collec-

ter les informations concernant le changement. Le type de l'entité de l'ontologie qui a changé, la caractéristique qui a changé et sa nouvelle valeur, la règle impactée et la portée de l'impact sont les informations nécessaires pour formuler un changement. Une fois ces informations collectées le type du changement est déterminé et l'instance du patron de changement correspondant est générée (voir table 1).

2. **Détection des incohérences**
   Une simulation de l'application du changement est effectuée pour analyser ses impacts sur l'ensemble des règles métier. Cette étape permet d'analyser le changement afin de détecter d'éventuelles incohérences. Un changement peut générer zéro ou plusieurs incohérences. Après l'instanciation du patron de changement, un patron d'incohérence est exécuté en fonction de la contrainte à vérifier et de la partie de la règle qui cause l'incohérence (i.e. prémisse ou conclusion).

3. **Réparation des incohérences**
   A partir des incohérences détectées, des solutions de réparation sont proposées en utilisant les patrons de réparation. Chaque solution proposée représente une opération de changement à appliquer. Il est donc impératif de s'assurer qu'une réparation ne cause pas d'autres incohérences. Le mécanisme de gestion de la cohérence est déclenché de manière récursif et seules les réparations qui ne causent pas d'incohérences sont retenues.
   Dans l'état actuel de l'avancement du travail, un patron de réparation génère un rapport de cohérence décrivant le changement effectué, le type de l'incohérence générée et le nom de la règle qui a généré l'incohérence.

3.2.2. Patrons de Gestion des Changements

Les **Patrons de Gestion des Changements** permettent de gérer de manière automatique le processus de gestion de la cohérence des règles métier lors de l'évolution de l'ontologie correspondante. Ils permettent de modéliser les invariances observées dans le processus de gestion de la cohérence. Nous définissons trois catégories de patrons de gestion des changements : les **Patrons de changement**, les **Patrons d'incohérence** et les **Patrons de réparation**.

1. **Patron de Changement**

   Il permet de modéliser le changement qui a affecté l'ontologie et la portée du changement sur les règles. Un patron de changement est dans une ontologie OWL et décrit :
   - l'entité ontologique concernée : concept ou propriété ;
   - la caractéristique impactée : l'identifiant de l'entité, la restriction sur l'entité, la hiérarchie de l'entité ... (Djedidi, 2009) ;
   - la nouvelle valeur de la caractéristique ;
   - les règles impactées par ce changement ;
   - la portée de l'impact du changement : impact au niveau de la prémisse ou de la conclusion d'une règle ;
   - le type de changement.

   Une instance du patron du changement correspond à un individu de l'ontologie de changement.

   Exemple 1. Soit le patron de changement correspondant au changement du domaine d'une restriction sur une propriété de type de donnée. Ce patron est instancié comme suit(voir Table 1).

P-Changement : Restriction-Domaine-Propriété	
Entité concernée	propriété de type de donné : `unitéTemps`
Caractéristique impactée	restriction : `allValuesFrom {"TLJ","A", "B", "C"}`
Nouvelle caractéristique	`allValuesFrom {"A", "B", "C"}`
Règle(s) impactée(s)	$r2$
Portée de l'impact	conclusion
Type changement	Restriction-Domaine-Propriété

TABLE 1: Exemple d'instance du patron de changement

2. **Patron d'Incohérence**

   Il permet de détecter le(s) type(s) d'incohérence(s) dû(s) à un changement. Un patron d'incohérence est modélisé sous forme de règles [6]. En

6. Afin d'éviter la confusion, il est important de distinguer entre les deux types de règles décrites dans cet article. Il y a les règles éditées par les utilisateurs métier, à partir d'une ontologie de domaine, sur lesquelles nous détectons les incohérences et les règles de gestion des incohérences qui composent le module de gestion de la cohérence, qui sont éditées à partir

fonction des informations contenues dans le patron de changement instancié, les règles modélisant les incohérences sont exécutées. Un changement peut ne causer aucune incohérence. Un patron d'incohérence est décrit par :

- les contraintes que le changement effectué doit vérifier ;
- localisation de l'incohérence : la prémisse ou la conclusion de la règle ;
- le type de l'incohérence.

En fonction de l'ontologie des changements, nous éditons les règles pour la gestion de la cohérence en utilisant le prototype développé pour l'édition des règles métier à partir d'ontologies (voir section 2.). Ci-dessous, deux exemples de règles l'une permettant de détecter une violation de domaine (RVD) et l'autre permet de détecter une règle jamais applicable (RJA).

```
RVD : si la nouvelle valeur de la propriété de la partie
      action de la règle est supérieure à la valeur max
      du domaine de la propriété de la règle
      alors l'incohérence est une violation de domaine;
```

```
RJA : si la nouvelle valeur de la propriété de la partie
      condition de la règle est inférieur à la valeur max
      du domaine de la propriété de la règle
      alors l'incohérence est règle jamais applicable;
```

Exemple2. En se basant sur le changement décrit dans la table 1, le patron d'incohérence exécuté est le suivant :

```
R-I : si la nouvelle valeur de la propriété de la partie
      action de la règle n'est pas parmi les litteraux du
      domaine de la propriété de la règle
      alors l'incohérence est violation de domaine;
```

3. **Patron de Réparation**

   Il permet de réparer une incohérence. Une ou plusieurs réparations peuvent être suggérées. Un patron de réparation est aussi modélisé sous forme

---

de l'ontologie de changement. Les deux types de règles sont éditées dans un langage naturel contrôlé. En d'autre termes, nous développons des règles pour détecter des incohérences sur d'autre règles.

de règle. En fonction de l'incohérence détectée le patron de réparation correspondant est exécuté. Un patron de réparation est décrit par :

- la règle à réparer : la règle causant une incohérence ;
- la partie impactée : prémisse ou conclusion ;
- la contrainte à vérifier ;
- la réparation : une solution qui permet de résoudre l'incohérence (cette caractéristique n'est pas encore prise en compte).

Exemple3. Ci-dessous un exemple de patron de réparation correspondant à l'incohérence détectée par la règle `R-I` décrite précédemment.

```
RP : si l'incohérence est violation de domaine
        alors afficher "Une violation de domaine
        est détectée dans la partie action de la
        regle" + le nom de la règle + " causée
        par la valeur assigné à la propriété" +
        le nom de la propriété;
```

L'exécution des règles modélisant les incohérences et les réparations est effectuée à travers un `rule flow`[7] qui permet de séquencer l'exécution des règles.

4. Résultats

Le but du travail effectué est de permettre à des utilisateurs métier d'éditer des règles métier en langage naturel contrôlé à partir d'une ontologie OWL et de gérer la cohérence de l'ensemble de ces règles en particulier lors de l'évolution de l'ontologie. Pour cela, nous avons développé deux prototypes, le `Plug-in OWL` et le `Consistency maintenance Plug-in`, sous forme de plug-in Eclipse pour JRules (Korf *et al.*, 2010).

Le `Plug-in OWL` permet l'édition des règles métier en langage naturel contrôlé à partir d'ontologies OWL en rendant possible l'importation des ontologies dans JRules.

Le `Consistency maintenance Plug-in` permet de gérer la cohérence d'un ensemble de règles métier éditées à partir d'une ontologie OWL, lors de l'évolution de cette ontologie (voir section 3.).

7. http ://publib.boulder.ibm.com/infocenter/brjrules/v7r1/index.jsp ?topic=/com.ibm.websphere.ilog.jrules.doc/html/api/html/ilog/rules/studio/model/ruleflow/package-summary.html

Pour expérimenter les prototypes développés, nous avons utilisé des ontologies OWL fournies par Audi [8] et Arcelor Mittal [9] qui sont nos partenaires sur le projet ONTORULE [10]. Nous avons également développé une ontologie OWL en se basant sur un modèle UML modélisant un système de validation de prescriptions pharmaceutiques (voir figure 2) au sein de l'Hôpital Européen George Pompidou (HEGP). Nous avons ensuite importé l'ontologie dans JRules et édité une dizaine de règles métier, parmi elles les règles $r1$, $r2$ et $r3$ décrites ci-dessus et la règle $r4$ ci-dessous. Pour éditer nos règles nous nous sommes fondés sur des règles qui ont été développées par (Boussadi *et al.*, 2011) au sein de l'HEGP en collaboration avec un utilisateur métier.

```
r4 : si l'unité de dosage du schema posologique de la prescri-
     ption du patient est "COMPRIME(S)"  et l'unité de temps du
     schéma posologique de la prescription du patient est "TLJ"
     alors assigner l'avis du medecin prescripteur de la pres-
     cription  à "Prescription invalide";
```

Nous avons ensuite simulé un changement sur l'ontologie (voir section 3.2.) qui consiste à changer le domaine de la restriction de la propriété `unitéTemps` de `allValueFrom {"TLJ","A", "B", "C"}` vers `allValueFrom {"A", "B", "C"}`. Le prototype analyse l'impact du changement sur l'ensemble des règles avant de l'appliquer. Le changement simulé impacte deux règles $r2$ et $r4$ et génère l'instance de patron de changement décrite dans la table 1 et une deuxième instance décrite dans la table 2.

Suite à l'analyse du changement, deux instances de changement ont été générées. Ces deux instances ont permis de déclencher deux règles au niveau du patron d'incohérence. Le premier type d'incohérence détecté est la violation de domaine, causé par la règle $r2$ qui assigne la valeur « TLJ » à la propriété `unitéTemps`. Cette incohérence a été détectée par la règle `R-I` du patron d'incohérence, décrite dans la section 3.2.2.. Le deuxième type d'incohérence causé par ce changement est détecté au niveau de la règle $r4$ qui ne sera plus applicable une fois le changement appliqué. Cette incohérence a été détecté par la règle `RJA` décrite dans la section 3.2.2.

8. Démonstration du plug-in OWL avec l'ontologie fournie par Audi : http ://ontorule-project.eu/deliverable-videos/d21-audi-ibm-video

9. Démonstration du plug-in OWL avec l'ontologie fournie par Arcelor Mittal : http ://ontorule-project.eu/deliverable-videos/d21-am-ibm-video

10. http ://ontorule-project.eu/

Entité concernée	propriété de type de donné : `unitéTemps`
Caractéristique impactée	restriction : `allValueFrom {"TLJ","A", "B", "C"}`
Nouvelle caractéristique	`allValueFrom {"A", "B", "C"}`
Règle(s) impactée(s)	$r4$
Portée de l'impact	prémisse
Type changement	Règle-jamais-applicable

TABLE 2: Exemple2 d'instance du patron de changement

5. Discussion et travaux existants

Pour développer notre approche de gestion de la cohérence nous nous sommes basés sur ONTO-EVO^{A}L, une méthode de gestion de la cohérence d'une ontologie lors de son évolution qui s'appuie sur une modélisation à base de patrons de gestion de changement. Cette méthode permet de gérer la cohérence d'un modèle conceptuel (une ontologie) lors de son évolution. L'approche que nous proposons permet la gestion de l'évolution et de la cohérence d'un modèle décisionnel (les règles métier) lors de l'évolution du modèle conceptuel correspondant.

Des travaux sur l'intégration des ontologies et des règles métier ont été effectués particulièrement dans le cadre du web sémantique. Pour permettre l'interaction entre ces deux formalismes, différents langages ont été développés tels que Semantic Web Rule Language (SWRL) [11], dl-programs (Eiter *et al.*, 2004), DL+log (Rosati, 2006), Minimal Knowledge and Negation as Failure (MKNF) (Motik & Rosati, 2007) et Autoepistemic Logic (FO-AEL) (de Bruijn *et al.*, 2007) ... Toutes ces approches sont fondées sur des règles écrites dans des langages techniques qui ne peuvent pas être assimilés par les utilisateurs métier. L'avantage du prototype développé est qu'il permet l'édition des règles métier, par l'utilisateur métier, dans un langage naturel contrôlé.

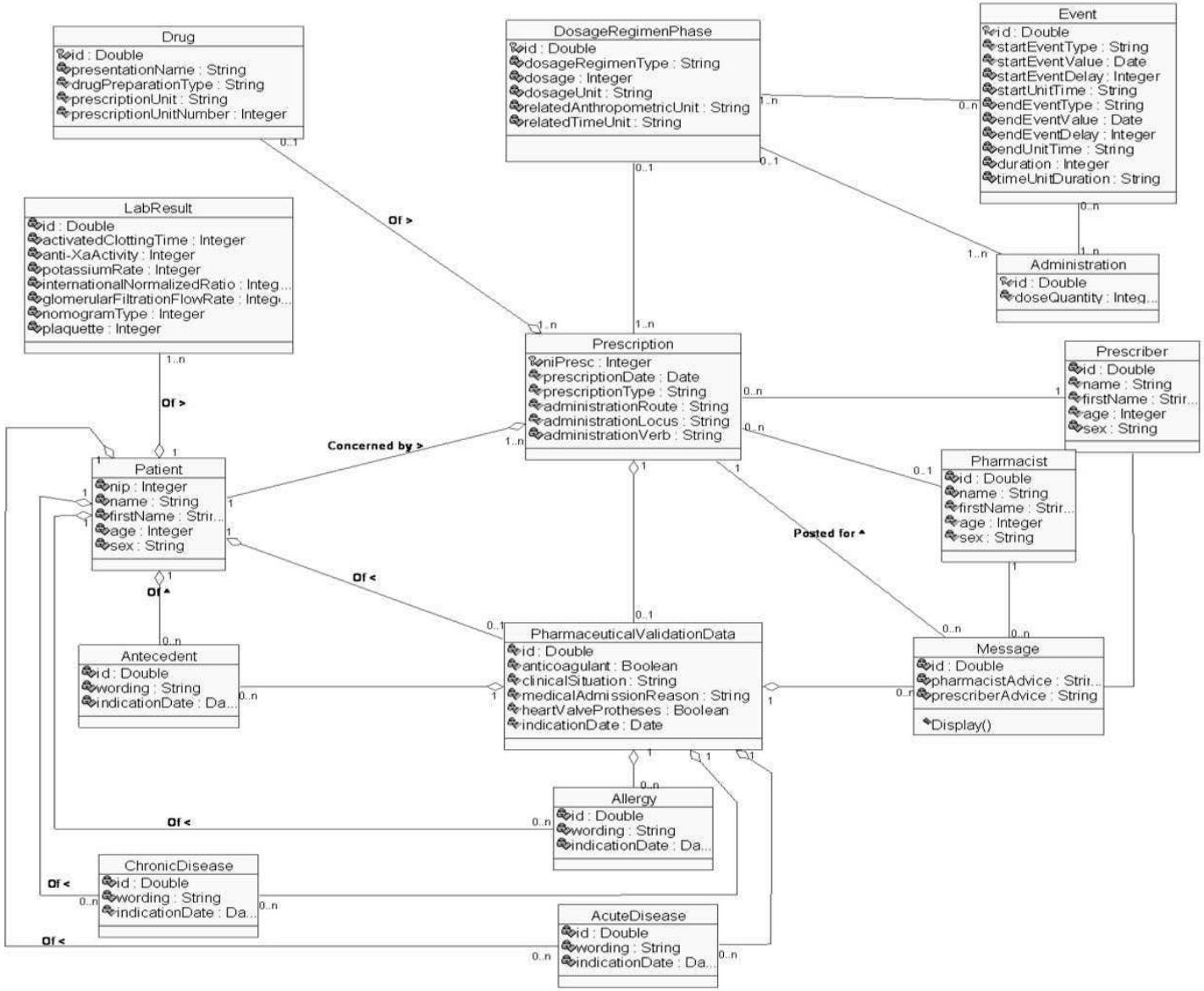
6. Conclusion et perspectives

Dans cet article, nous avons décrit la méthode développée pour la gestion de la cohérence des règles métier éditées à partir d'ontologies écrites en OWL

11. http ://www.w3.org/Submission/SWRL/

lors de l'évolution de cette dernière. Pour tester notre approche, nous avons utilisé des ontologies fournies par nos partenaires dans le projet ONTORULE, et nous avons aussi développé une ontologie modélisant un système de validation de prescription pharmaceutique en se basant sur le diagramme UML décrit dans la figure 2. En utilisant le `plug-in OWL`, nous avons importé cette ontologie dans JRules et édité des règles métier en langage naturel contrôlé (voir section 3.). Nous avons ensuite simulé un changement sur l'ontologie et détecté les incohérences causées par ce changement au niveau des règles en utilisant le `Consistency Management Plug-in` (voir section 4.).

A l'état actuel du travail, les incohérences détectées sont la violation de domaine et les règles qui ne sont jamais applicables. En perspective, nous envisageons de détecter les incohérences citées dans la section 3.1. et d'améliorer les patrons de réparation afin de proposer des solutions aux incohérences détectées et guider l'utilisateur dans le processus de réparation des incohérences. Pour cela, nous allons nous baser sur la méthode CommonKADS (Schreiber *et al.*, 2000) qui définit un modèle de connaissance en trois catégories, `les connaissances du domaine`, `les connaissances d'inférence` et `les connaissances de la tâche`. `Les connaissances du domaine` représentent les connaissances spécifiques d'un domaine et les types ses informations tels que par example les types de changement et leurs caractéristiques, leurs impactes sur les règles et les incohérences potentiellement générées. `Les connaissances d'inférence` représentent les étapes d'inférences à effectuer en fonction des connaissances du domaine. Par example, en fonction d'un ensemble de règles et du domaine de connaissance, les règles de gestion de la cohérence, correspondant au patron d'incohérence, sont déclencher pour détecter les incohérences causées par une changement. Enfin, `les connaissances de la tâche` qui représentent le but à atteindre et comment l'atteindre en décrivant le processus à suivre qui dans notre cas représentent les propositions de réparation des incohérences détectées et comment appliquer ces réparations.


Drug
id : Double
presentationName : String
drugPreparationType : String
prescriptionUnit : String
prescriptionUnitNumber : Integer
DosageRegimenPhase
id : Double
dosageRegimenType : String
dosage : Integer
dosageUnit : String
relatedAnthropometricUnit : String
relatedTimeUnit : String
Event
id : Double
startEventType : String
startEventValue : Date
startEventDelay : Integer
startUnitTime : String
endEventType : String
endEventValue : Date
endEventDelay : Integer
endUnitTime : String
duration : Integer
timeUnitDuration : String
LabResult
id : Double
activatedClottingTime : Integer
anti-XaActivity : Integer
potassiumRate : Integer
internationalNormalizedRatio : Integ...
glomerularFiltrationFlowRate : Integ...
nomogramType : Integer
plaquette : Integer
Administration
id : Double
doseQuantity : Integ...
Prescription
niPresc : Integer
prescriptionDate : Date
prescriptionType : String
administrationRoute : String
administrationLocus : String
administrationVerb : String
Prescriber
id : Double
name : String
firstName : Strir...
age : Integer
sex : String
Pharmacist
id : Double
name : String
firstName : Strir...
age : Integer
sex : String
Patient
nip : Integer
name : String
firstName : Strir...
age : Integer
sex : String
Antecedent
id : Double
wording : String
indicationDate : Da...
PharmaceuticalValidationData
id : Double
anticoagulant : Boolean
clinicalSituation : String
medicalAdmissionReason : String
heartValveProtheses : Boolean
indicationDate : Date
Message
id : Double
pharmacistAdvice : Strir...
prescriberAdvice : String
Display()
Allergy
id : Double
wording : String
indicationDate : Da...
ChronicDisease
id : Double
wording : String
indicationDate : Da...
AcuteDisease
id : Double
wording : String
indicationDate : Da...
Of >
Concerned by >
Of <
Posted for ^
Of ^
Of <
Of <
Of <

Remerciements

Nous tenons à remercier Abdellali Boussadi et Patrice Degoulet pour nous avoir fournit les informations et les données nécessaires pour tester notre travail. Nous remercions aussi Adil El Ghali, Philippe Bonnard et Hugues Citeau pour leurs relectures et leurs contributions pour améliorer l'article.

Références

ANTONIOU G., DAMASIO C. V., GROSOF B., HORROCKS I., KIFER M., MALUSZYNSKI J. & PATEL-SCHNEIDER. P. F. (2004). Combining rules and ontologies : A survey. *Technical Report IST506779/Link¨oping/I3-D3/D/PU/a1, Link¨oping University*. http ://rewerse.net/publications/.

BOUSSADI A., BOUSQUET C., SABATIER B., CARUBA T., DURIEUX P. & DEGOULET P. (2011). A business rules design framework for a pharmaceutical validation and alert system. *Methods of Information in Medicine*.

CHNITI A., DEHORS S., ALBERT P. & CHARLET J. (2010). Authoring business rules grounded in owl ontologies. In M. D. ET AL. (EDS.), Ed., *RuleML 2010 : The 4th International Web Rule Symposium : Research Based and Industry Focused* : LNCS 6403, Springer-Verlag Berlin Heidelberg 2010.

DE BRUIJN J., EITER T., POLLERES A. & TOMPITS H. (2007). Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. *IJCAI*.

DJEDIDI R. (2009). *Approche d'évolution d'ontologie guidée par des patrons de gestion de changement*. PhD thesis, Université Paris-Sud XI Orsay.

DJEDIDI R. & AUFAURE M. (2010). Onto-evoal an ontology evolution approach guided by pattern modelling and quality evaluation. *Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2010)*.

EITER T., IANNI G., POLLERES A., SCHINDLAUER R. & TOMPITS H. (2006). Reasoning with rules and ontologies. *Reasoning Web 2006*, p. 93–127.

EITER T., LUKASIEWICZ T., SCHINDLAUER R. & TOMPITS H. (2004). Combining answer set programming with description logics for the semantic web. *KR*.

KALYANPUR A., JIMENEZ D., BATTLE S. & PADGET J. (2004a). Automatic mapping of owl ontologies into java. *Frank Maurer and GÄunther Ruhe*

(eds) Proc. of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004).

KALYANPUR A., JIMENEZ D., BATTLE S. & PADGET J. (2004b). Detailed technical report on mapping owl to java.

KORF R., KISS E., DURAND F., DOUSEK M., EL GHALI A., CHNITI A., VIGUIÉ S., BERRUETA D. & LÉVY F. (2010). D2.4 : Extended demonstration prototypes for dissemination, teaching and public experimentation purposes. *ONTORULE Delivrable, http ://ontorule-project.eu/deliverables.*

LEONE N., GOTTLOB G., ROSATI R., EITER T., FABER W., FINK M., GRECO G., IANNI G., KALKA E., LEMBO D., LENZERINI M., LIO V., NOWICKI B., RUZZI M., STANISZKIS W. & TERRACINA. G. (2006). The infomix system for advanced integration of incomplete and inconsistent data. *Reasoning Web 2006*, p. 93–127.

MOTIK B. & ROSATI R. (2007). A faithful integration of description logics with logic programming. *IJCAI*.

PÜHRER J., EL GHALI A., CHNITI A., KORF R., SCHWICHTENBERG A., LÉVY F., HEYMANS S., XIAO G. & EITER T. (2010). D2.3 consistency maintenance. intermediate report. *ONTORULE Delivrable, http ://ontorule-project.eu/deliverables.*

ROSATI R. (2006). Dl+log : Tight integration of description logics and disjunctive datalog. *KR*.

SCHREIBER G., AKKERMANS H., ANJEWIERDEN A., DE HOOG R., SHADBOLT N., DE VELDE W. V. & WIELINGA B. (2000). *Knowledge Engineering and Management : The CommonKADS Methodology*.

STOJANOVIC L. (2004). *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe.