HAL
open science

Représentation et vérification d'un environnement intelligent à partir de spécifications utilisateur en langage naturel

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane, Brigitte Grau

To cite this version:

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane, Brigitte Grau. Représentation et vérification d'un environnement intelligent à partir de spécifications utilisateur en langage naturel. Revue des Sciences et Technologies de l'Information - Série RIA : Revue d'Intelligence Artificielle, 2015, 29 (1), pp.47-81. 10.3166/RIA.29.47-81 . hal-01378061

HAL Id: hal-01378061

https://hal.science/hal-01378061v1

Submitted on 22 Oct 2019

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.

Représentation et vérification d'un environnement intelligent à partir de spécifications utilisateur en langage naturel

Driss Sadoun [1], **Catherine Dubois** [2,4], **Yacine Ghamri-Doudane** [3], **Brigitte Grau** [1,4]

1. LIMSI-CNRS, Université Paris-Sud, France *2. CEDRIC/CNAM, France*
3. L3i Lab, Université de La Rochelle, France *4. ENSIIE, France*

RÉSUMÉ. *Aujourd'hui des capteurs et actionneurs associés à des périphériques de contrôle peuvent être installés n'importe où, notamment dans nos maisons, créant des environnements intelligents. Notre objectif est de permettre à un utilisateur de configurer son propre environnement intelligent en décrivant ses besoins, i.e. les règles de comportement de l'environnement, en langage naturel (LN). Nous explorons les possibilités offertes par une ontologie formelle pour faire le lien entre spécifications en LN et spécifications formelles. L'analyse des spécifications LN permet l'instanciation automatique de l'ontologie afin qu'elle représente le comportement décrit par l'utilisateur. Les règles de comportement représentées sont alors traduites en spécifications Maude, afin de compléter les vérifications possibles sous OWL. Nous montrons que tout au long de ce processus de formalisation, il est possible de vérifier la complétude, la cohérence et la conformité des exigences spécifiées et de maintenir une traçabilité entre spécification LN et spécifications formelles autorisant un retour précis à l'utilisateur.*

ABSTRACT. *Nowadays sensors and actuators associated with control devices can be installed anywhere, as in our homes creating smart environments. Our goal is to allow a user to configure her own smart environment by describing her needs, i.e. the environment behavioral rules, in natural language (NL). We explore the possibilities offered by an ontology, to transform NL specifications into formal specifications. Analysis of user requirements allows us an automatic instantiation of the ontology so that it represents the behavior described by the user. The represented behavioral rules are then translated into Maude specifications to complement verifications realized in OWL. We show that throughout this formalization process, it is possible to check the completeness, the consistency and the conformity of the specified requirements and maintain traceability between NL requirements and formal specifications to allow a precise feedback to the user.*

MOTS-CLÉS : *environnement intelligent, ontologie, spécifications, vérification formelle.*

KEYWORDS: *smart environment, ontology, natural language requirements, formal verification.*

---

1. Introduction

1.1. Motivation

Nous assistons aujourd'hui à un intérêt accru envers l'utilisation des nouvelles technologies de l'information et de la communication pour résoudre des problèmes de la vie courante. L'apparition de l'informatique ubiquitaire et le développement des dispositifs de capture miniaturisés et des communications machines à machines font, qu'il devient réaliste d'imaginer des applications auxquelles nous n'aurions jamais pensé auparavant. Aujourd'hui des capteurs et actionneurs associés à des périphériques de contrôle peuvent être installés n'importe où : dans nos maisons, nos immeubles, nos bureaux et nos villes, créant des environnements intelligents. Cependant, ces environnements intelligents ne sont pas aisément adaptables aux besoins de l'utilisateur, ce qui limite leur déploiement.

En effet, un utilisateur qui installerait un système domotique au sein de son logement n'aurait aujourd'hui que peu de possibilité de voir ce système s'adapter à ses exigences. La configuration de ce système domotique sera alors statique ou quasi-statique. Or, les capteurs et actionneurs installés dans cet environnement intelligent pourraient permettre une pléiade d'utilisation possible. Ainsi, à titre d'exemple, un capteur de température dans une pièce pourrait permettre à la fois de piloter le chauffage et le système d'air conditionné de l'environnement mais aussi de détecter des sources de déperdition d'énergie, par exemple une fenêtre ouverte. Aujourd'hui un système domestique classique ne servirait qu'à une voire deux utilisations seulement par capteur et permettrait rarement de raisonner à partir de données issues de plusieurs capteurs pour identifier une situation. La puissance que pourrait offrir un tel système n'est donc pas exploitée. C'est cette puissance que l'on obtiendrait si l'environnement intelligent était adaptable, ce qui permettrait de dépasser les limites actuelles de déploiement. Ce raisonnement appliqué aux systèmes domotiques ci-dessus s'applique également à tout type d'environnement intelligent.

De nombreux outils apparaissent aujourd'hui et permettent d'effectuer la réalisation d'une spécification énoncée d'un environnement intelligent. Nous citerons à titre d'exemples les travaux de (Cherrier *et al.*, 2011 ; 2013 ; Reijers *et al.*, 2013). Or ces outils de configuration et de déploiement de configurations sont très complexes et ne sont pas accessibles à un utilisateur non expert. Ils doivent nécessairement faire appel à des compétences en informatique ce qui, là aussi, limite leur utilisation à des utilisateurs experts. Leur utilisation permet donc une adaptabilité du système mais pas de façon aisée.

Tout projet de développement logiciel complexe commence normalement par la spécification des exigences décrivant le système à développer. Dans nos travaux nous nous sommes attachés à répondre à la question : « comment rendre un environnement intelligent aisément adaptable ? ». Or, qu'est-ce qui serait plus aisé pour décrire une configuration que le langage naturel (LN) ? En effet, la majorité des spécifications sont écrites car celles-ci sont souvent contractuelles et spécifiées en langage naturel (Mich

et al., 2004) car ce dernier est de loin le moyen le plus simple pour un utilisateur d'exprimer ses besoins.

L'outil de configuration de l'environnement intelligent devra alors être en mesure de transformer des spécifications LN en une configuration des éléments composant l'environnement intelligent. Des outils tel que ceux cité ci-dessus (Cherrier *et al.*, 2011 ; 2013 ; Reijers *et al.*, 2013), pourront alors prendre le relais pour la réalisation effective de l'environnement intelligent spécifié par l'utilisateur. Avant cela, il faudrait être à même de représenter et surtout de valider la spécification énoncée en langage naturel par l'utilisateur. C'est cette représentation et cette validation qui constitue le propos de cet article. Dans ce travail [1], l'objectif est de démontrer la faisabilité du passage d'une spécification LN à une représentation valide, cette dernière sera par la suite le point de départ de la réalisation de la spécification d'un environnement intelligent (*i.e.* génération et déploiement de la configuration sur les capteurs, actionneurs et processus de contrôle). Cette génération ne sera pas abordée dans cet article. Ainsi, notre ambition est de permettre à un utilisateur [2] de configurer son propre environnement intelligent simplement en décrivant ses besoins, *i.e.* les règles de comportement de l'environnement, en langage naturel. Cela suppose d'être en mesure de produire un retour à l'utilisateur sur la qualité des spécifications d'exigences qu'il aura rédigées, à savoir si celle-ci sont complètes, cohérentes et conformes aux contraintes du domaine.

1.2. Contributions

Les spécifications d'exigences ont pour objet de décrire les caractéristiques du système à développer (son comportement, ses propriétés et contraintes), afin de guider et de faciliter sa réalisation. Pour être vérifiées, les spécifications doivent être représentées de telle sorte à pouvoir être traitées par des méthodes de vérification formelles qui permettent de démontrer leur validité. Ces méthodes permettent d'obtenir une très forte assurance quant à la cohérence des spécifications décrivant le système à réaliser. Toutefois, les spécifications en LN sont souvent porteuses de formulations ambiguës ou implicites pouvant mener à des problèmes d'interprétation et ne sont donc pas considérées comme formelles.

L'automatisation d'un passage direct entre spécifications LN et spécifications formelles est difficile, si ce n'est impossible à effectuer (Guissé *et al.*, 2012). Cette problématique n'est pas récente (Fraser *et al.*, 1991 ; Vadera, Meziane, 1994 ; Fougères, Trigano, 1997). Elle a suscité et suscite encore un grand intérêt (Bajwa *et al.*, 2012 ; Guissé, 2013) car elle donne lieu à de nombreux travaux dans différents domaines scientifiques tels que l'ingénierie des exigences, la représentation des connaissances, le traitement automatique de la langue ou la vérification formelle. L'ensemble de ces travaux a montré la nécessité de passer par une représentation pivot. En parti-

1. Nos travaux ont été financés par DIGITEO, projet DIM LSC 2010.

2. Dans cet article, nous ne faisons pas de distinction entre des spécifications écrites par un ou plusieurs utilisateurs.

culier, les recherches portant sur l'application de techniques d'IA à l'ingénierie des exigences ont connu une croissance très importante au cours des deux dernières décennies (Meziane, Vadera, 2010), notamment l'utilisation de modèle de représentation semi-formelle ou formelle pour l'aide à la spécification d'exigences (Körner, Brumm, 2009 ; Castañeda *et al*., 2012 ; Chniti *et al*., 2012 ; Al Balushi *et al*., 2013).

Il n'existe à ce jour pas de solution complète permettant une vérification formelle de spécifications écrites en langue naturelle. Notre objectif est donc de fournir une telle solution, qui permette de tracer et diagnostiquer les erreurs afin d'être en mesure d'effectuer un retour à l'utilisateur. Cela nous a amené à choisir un formalisme de représentation adapté à la fois pour l'analyse de textes en langue naturelle et à la génération de spécifications formelles.

Nous proposons d'explorer les possibilités offertes par un modèle de représentation fondé sur un formalisme logique, en l'occurrence une ontologie, pour jouer le rôle de modèle de représentation pivot et faire le lien entre spécifications en LN et spécifications formelles. L'originalité de notre proposition réside dans l'usage d'une ontologie comme modèle intermédiaire pour l'automatisation du passage et le maintien d'une traçabilité entre spécification LN et spécifications formelles. L'ontologie proposée est décrite en OWL-DL une version décidable du langage d'ontologie web OWL fondé sur les logiques de description. Elle fournit un cadre de représentation sémantique explicite qui permet de guider l'analyse d'exigences écrites en LN, lever des ambiguïtés liées à la langue et réaliser des inférences pour expliciter certaines informations. De plus, l'ontologie, de par son formalisme logique, permet d'effectuer un certain nombre de vérifications, telles que vérifier la cohérence des connaissances explicitées à l'aide de moteur d'inférence ou la complétude des connaissances par l'application de requêtes. Toutefois, les vérifications que nous souhaitons effectuer ne sont pas toutes possibles sous OWL. En effet, le raisonnement OWL est monotone [3] et s'applique de façon globale sur l'ensemble des instances et des règles de l'ontologie. Il ne permet donc pas de représenter les changements d'états ou différents ordres d'enchaînement séquentiels de règles de comportement, potentiellement concurrentes. À l'aide des moteurs d'inférences et de requêtes OWL, il est possible d'effectuer différents tests sur les connaissances représentées et les évolutions dynamiques de l'état du système. Néanmoins, OWL ne permet pas une vérification exhaustive et systématique de tous les états possibles d'un système. Ce dernier type de vérification est permis par une approche de type *model-checking*. Afin de compléter les vérifications d'OWL, le langage formel cible doit donc permettre l'application de techniques de model-checking.

Différents formalismes permettent l'exécution de techniques de model-checking. Parmi les plus largement utilisés on trouve les réseaux de Petri (Reisig, 1985). Néanmoins, notre choix s'est orienté vers le langage de spécification MAUDE (Clavel *et al*., 2011) car il permet d'exprimer différentes logiques (Martí-Oliet, Meseguer, 2002)

3. Lors d'un raisonnement monotone, ajouter une nouvelle connaissance ne fera qu'augmenter les connaissances de l'ontologie et ne pourra mener à la révision de connaissances.

et qu'il est donc bien adapté pour représenter des spécifications modélisées en logique de description sur laquelle est fondé OWL. Maude est fondé sur la logique de réécriture et permet de décrire l'état d'un système et son comportement. Il est bien adapté pour la spécification de systèmes concurrents (Romero *et al.*, 2007) et intègre une panoplie d'outils de vérification (Clavel *et al.*, 2007) dont un modèle checker (Eker *et al.*, 2004).

Nous proposons une ontologie de haut niveau, modélisant le comportement générique d'un environnement intelligent, de manière à disposer d'une modélisation du domaine qui soit instanciable quelle que soit la configuration physique du système (*i.e.* le lieu, sa constitution, les types de capteurs installés). L'analyse des spécifications utilisateur, guidée par les connaissances décrites dans l'ontologie, provoque l'instanciation automatique de celle-ci afin qu'elle représente le comportement décrit par l'utilisateur. Les règles de comportement qui y sont représentées sont ensuite traduites en spécifications Maude, afin de compléter les vérifications faites sous OWL. Nous montrons comment tout au long de ce processus de formalisation des spécifications d'exigences, il est possible de vérifier la complétude, la cohérence et la conformité des exigences énoncées et maintenir une traçabilité entre spécification utilisateur et spécifications formelles afin de permettre un retour précis à l'utilisateur.

1.3. Organisation de l'article

Dans cet article, la section 2 permet de nous positionner par rapport aux travaux connexes. En section 3, nous présentons les problèmes liés à la vérification de spécifications en LN et les solutions que nous y apportons. Nous décrivons ensuite, notre ontologie pivot en section 4 et son instanciation par les spécifications analysées en section 5. En section 6, nous décrivons le passage de l'ontologie pivot à Maude. Dans la section 7, nous présentons les résultats obtenus sur un corpus de test. Enfin, l'article se termine par une conclusion et la présentation des perspectives.

2. Travaux connexes

L'évolution des technologies ubiquitaires permet d'envisager des environnements intelligents pouvant s'adapter à leurs utilisateurs. Ces nouveaux systèmes qui peuvent interagir avec l'humain sont de plus en plus complexes. En fonction de leurs champs d'application, leur vérification peut être critique en termes de fiabilité ou de sécurité (Corno, Sanaullah, 2013) comme par exemple lorsqu'ils sont voués à s'adapter à des personnes handicapées (Guillet *et al.*, 2013). Lors d'un développement logiciel, la vérification peut intervenir à différents stades. Ainsi, pour vérifier si un environnement intelligent construit est bien celui attendu, on s'intéresse alors à son implémentation, tel que cela est fait dans (Yan *et al.*, 2008) ou (Du Bousquet *et al.*, 2009) qui présentent la validation d'une implémentation Java d'un réseau domestique reposant sur l'usage du *Java Modeling Language* (*JML*) et de *JUnit*. Toutefois, à ce stade de développement les erreurs identifiées sont bien plus coûteuses que si elles avaient été identifiées plus tôt dans le cycle de développement. Dans (Corno, Sanaullah, 2013) ou (Guillet

et al., 2013) la vérification porte sur une modélisation d'un environnement intelligent. Les erreurs identifiées permettent alors de fixer les bugs liés au modèle. Néanmoins, aucun retour n'est fait à l'utilisateur afin qu'il puisse revoir la spécification de ses exigences. Dans ce cas, aucune vérification du passage entre spécifications et modèles n'est possible. À notre connaissance, il n'existe pas d'approches automatiques permettant la vérification et l'amélioration de spécifications d'exigences formulées en LN décrivant le comportement d'un environnement intelligent.

Le principal verrou lors du passage de spécifications en LN vers des spécifications formelles réside dans la distance entre les deux formalismes. Aussi la question a été abordée par la proposition de formalismes intermédiaires visant à réduire cette distance. La problématique est alors de choisir une représentation suffisamment formelle et précise pour permettre un passage automatisé vers des spécifications formelles et suffisamment expressive pour représenter la sémantique des connaissances formulées en LN. Dans la littérature on trouve le recours aux langages naturels contrôlés tel que SBVR (Njonko, El Abed, 2012 ; Guissé *et al.*, 2012) ou l'utilisation de modèles de représentation semi-formels tel que UML (Bajwa *et al.*, 2012). L'inconvénient de ces deux formalismes est qu'ils manquent de mécanismes de validation et de moteurs d'inférence. Ces manques ont amené plusieurs chercheurs à explorer les avantages d'une transformation de règles SBVR vers des langages dotés de mécanismes d'inférence et de validation, tels que OWL et SWRL (Karpovic, Nemuraite, 2011 ; Sukys *et al.*, 2012) ou de modèle UML vers des langages de spécifications formelles, notamment vers le langage Maude (Roldan, Duran, 2010 ; Duràn, Roldàn, 2012).

Le recours aux modèles formels tels que les ontologies comme représentation intermédiaire n'a pas encore été exploré. Pourtant, les travaux sur le passage automatique de textes en LN vers des ontologies ont montré des résultats prometteurs (Petasis *et al.*, 2011 ; Ruiz-Martìnez *et al.*, 2011 ; Thongkrau, Lalitrojwong, 2012) et divers travaux ont déjà proposé des méthodes de traduction automatique entre une ontologie et un langage formel tels que Alloy (Song *et al.*, 2012) ou Maude (Verdejo *et al.*, 2005 ; Senanayake *et al.*, 2005 ; Huang *et al.*, 2009). En outre, de nombreux travaux insistent sur la pertinence de l'usage d'ontologies en ingénierie des exigences (Castañeda *et al.*, 2010), pour codifier les connaissances pertinentes et faciliter la compréhension du domaine (Omoronyia *et al.*, 2010 ; Al Balushi *et al.*, 2013), minimiser l'ambiguïté linguistique (Castañeda *et al.*, 2012), identifier des règles métier redondantes, jamais applicables ou qui violent le domaine de leur définition (Chniti *et al.*, 2012), ou s'appuyer sur les connaissances d'ontologies pour aider l'analyse dans ses décisions pour améliorer des spécifications (Körner, Brumm, 2009). Enfin, l'usage d'ontologies pour la modélisation d'environnements intelligents est recommandé par divers auteurs (Xu *et al.*, 2009 ; Wei *et al.*, 2012 ; Bae, 2014 ; Li *et al.*, 2014). Nous proposons donc d'étudier et d'exploiter l'apport d'une ontologie comme représentation pivot dans un processus de validation de spécifications en LN décrivant le comportement d'un environnement intelligent.

3. Vérification des spécifications utilisateur : problèmes et solutions

3.1. Environnement intelligent et spécifications utilisateur

Un environnement intelligent est un ensemble d'objets communicants (capteurs, actionneurs et processus de contrôle) pouvant influencer le comportement des équipements auxquels ils sont reliés, sous des conditions bien définies. Un *capteur*, ou *détecteur*, est un appareil de détection ou de mesure dans une zone restreinte, sensible à différents types de phénomènes. Il est destiné à émettre un signal lorsqu'il détecte l'occurrence d'un phénomène ou à quantifier une grandeur physique. Un *actionneur* est un dispositif connecté à un appareil (processus physique) de l'environnement qu'il peut actionner. L'actionneur entre en activité lorsqu'il reçoit un signal particulier émanant d'un capteur. En fonction du signal qu'il reçoit, il peut effectuer une ou plusieurs actions (*allumer*, *ouvrir*, *éteindre* ...) sur les processus physiques qu'ils contrôlent. Les *processus de contrôle* définissent les règles de communication entre les différents dispositifs de l'environnement intelligent.

Le but est de permettre à un utilisateur non expert de décrire le comportement d'un environnement intelligent, dans son espace personnel de vie ou de travail. La qualité de ses spécifications doit être évalués afin de lui faire un retour précis et compréhensible du résultat, en cas de problème. Mais, pour être vérifiées, les spécifications utilisateur doivent être représentées de manière formelle. Un langage formel a des fondements mathématiques qui le rendent non ambigu et interprétable par les machines. Néanmoins, il n'est pas adapté à l'utilisateur du système car il est complexe et son utilisation requiert un certain degré d'expertise. Cet utilisateur étant impliqué dans la spécification des exigences et la validation de leur forme finale, l'emploi du langage naturel s'impose, car il ne nécessite aucun effort d'adaptation de sa part, contrairement à l'usage d'une interface, d'un vocabulaire restreint ou d'une syntaxe contrôlée.

L'inconvénient des spécifications en LN est qu'elles contiennent souvent des formulations ambiguës ou implicites, pouvant mener à des problèmes d'interprétation. Les spécifications suivantes, décrivent trois règles utilisateur [4] : *When I **enter** a **room** the **door opens** automatically. Each time the **window is open** in a **room shut** its **doors**. If a **movement is detected** in the **bedroom**, **turn on** the **light**.* . On remarque que les termes en gras dans les spécifications dénotent des entités ou relations du domaine des environnements intelligents : des phénomènes (*enter* et *movement is detected*), des localisations (*room* et *bedroom*), des processus physiques (*door* et *light*) ainsi que des propriétés (*open* et *turn on*). On s'aperçoit aussi que les mentions aux capteurs et actionneurs y sont implicites. De plus, on voit que certaines formulations telles que *turn on the light* peuvent mener à diverses interprétations, car le sens des termes *turn*, *on* et *light* sont multiples et différents en fonction de leur contexte d'énonciation. Enfin, on remarque que les deux premières phrases décrivent des phénomènes pouvant

4. Acquises via la plate-forme (http://perso.limsi.fr/sadoun/Application/fr/SmartHome.php) permettant de spécifier le comportement d'un environnement intelligent décrit de manière textuelle et graphique.

apparaître simultanément et qui débouchent sur des actions opposées (*door opens* et *shut its door*), et peuvent mener à une incohérence.

Pour permettre à un utilisateur de décrire le comportement de son environnement intelligent sans contrainte, nous proposons l'emploi itératif des langages naturel et formel. Il est donc essentiel de maintenir une traçabilité et de gérer les transitions de l'un à l'autre. La figure 1 illustre l'ensemble du processus, allant de la modélisation de l'ontologie pivot (1), instanciée à partir de l'analyse des spécifications (2) puis traduite en spécifications formelles Maude (3), avec un retour utilisateur issu des vérifications effectués sous OWL et Maude (4) qui, s'il est positif, mène à la réalisation du système sinon à la révision des spécifications par l'utilisateur.

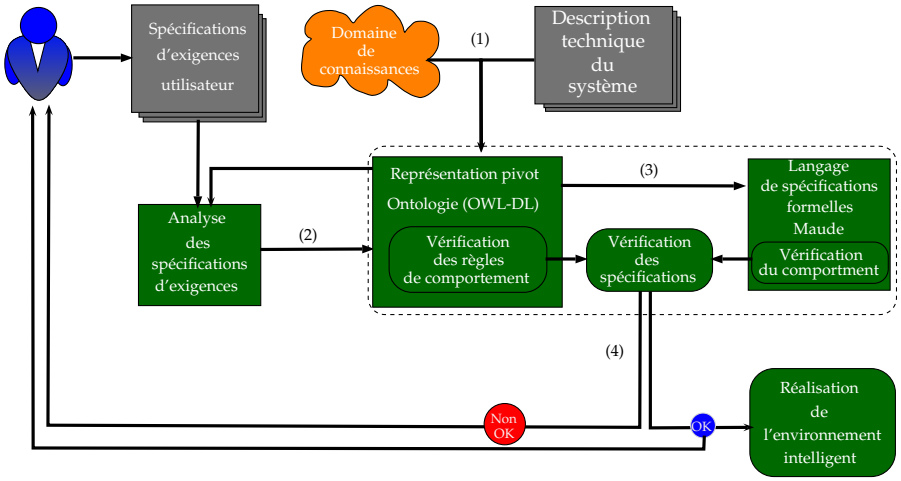

Figure 1. Processus de validation des spécifications et de retour utilisateur

Valider la qualité des spécifications décrites par un utilisateur fait appel à des mécanismes de vérification propres aux outils associés aux formalismes utilisés. Il s'agit alors d'exploiter les outils adéquats pour chaque type de vérification à effectuer.

3.2. Définition des vérifications effectuées

L'évaluation des spécifications utilisateur consiste à vérifier les règles de comportement de l'environnement intelligent construites à partir des textes. Ces règles sont des formules logiques, constituées d'un antécédent et d'un conséquent, eux-mêmes composés de prédicats. Un prédicat peut porter sur une ou plusieurs entités du domaine. Nous proposons de vérifier les règles énoncées selon trois aspects.

3.2.1. Applicabilité de la règle

Une règle spécifiée par un utilisateur doit pouvoir s'appliquer dans la pratique. Vérifier que l'application d'une règle est possible revient à vérifier la satisfaisabilité

de ses prédicats. Cela dépend en grande partie de la configuration physique de l'environnement intelligent, c'est-à-dire des caractéristiques de ses composants et de leurs interactions possibles. L'utilisateur peut, par exemple, souhaiter une interaction entre deux composants qui, dans la pratique ne sont pas intégrés dans l'environnement ou ne peuvent interagir, *i.e.* qui ne sont pas physiquement reliés. Dans ce cas l'utilisateur doit être averti afin de réviser ses exigences ou veiller à ce que la configuration physique de l'environnement intelligent évolue. Cette vérification est possible sous OWL à l'aide de comparaisons ensemblistes entre antécédent et conséquent d'une règle et les résultats de requêtes SQWRL (O'Connor, Das, 2008) sur les connaissances de l'ontologie.

3.2.2. Cohérence de la règle

La cohérence d'une règle de comportement est établie par rapport à l'ensemble des autres règles. Une règle est incohérente si son conséquent est en contradiction avec celui d'une autre règle pouvant s'appliquer simultanément. Deux cas se présentent. Le premier, qui est le plus simple à identifier, est lorsque deux règles en contradiction ont un même antécédent. Sous OWL cela correspond à une comparaison ensembliste sur l'ensemble des antécédents et conséquents des règles représentées. Dans le second cas, les antécédents sont différents mais peuvent être satisfaits en même temps. L'identification de ce cas n'est pas triviale et nécessite l'exploration de l'ensemble des applications possibles des règles de l'environnement intelligent. Cette vérification fait appel aux techniques de réécriture de Maude.

3.2.3. Conformité des règles

Les règles sont dites conformes si le comportement qu'elles décrivent répond d'une part aux exigences spécifiées par l'utilisateur et d'autre part aux contraintes du domaine. Dans le premier cas, elles sont issues des spécifications d'exigences et doivent répondre aux besoins utilisateur : ne pas dépasser un seuil de consommation énergétique. Dans le second cas, elles doivent être spécifiées par un spécialiste du domaine, donc en *Maude* dans notre approche, et répondre à des contraintes indépendantes des utilisateurs qui découlent du bon sens : tous les dispositifs de l'environnement doivent pouvoir être allumés et être éteints.

La vérification de la conformité des règles consiste à vérifier que l'ensemble des règles permet qu'à partir d'une configuration donnée, tous les états de l'environnement intelligent devant être atteints le soient et qu'aucun état indésirable ne soit jamais atteint. Par exemple, que chaque appareil de l'environnement puisse être allumé et que deux actions contradictoires [5] (allumer-éteindre, augmenter-diminuer, ...) ne puissent s'appliquer simultanément à un même dispositif (ampoule, chauffage, ...). Ce type de vérification fait appel aux techniques de réécriture de Maude.

5. Cette information est modélisée au sein de l'ontologie.

La liaison entre les résultats des différentes vérifications effectuées sous OWL et Maude avec les spécifications utilisateur est maintenue à l'aide de l'ontologie décrite dans la section suivante.

4. Description de l'ontologie

L'objectif de l'ontologie est triple : 1) Elle doit permettre de décrire de manière générique tout environnement intelligent et pouvoir être instanciée en fonction des spécifications d'exigences analysée ; 2) permettre de guider l'analyse des textes. Nous avons donc choisi d'axer la caractérisation des concepts par leur propriétés, de manière à disposer de contexte d'interprétation de leurs instances lors de l'analyse des textes; 3) autoriser une traduction automatique en Maude, donc modéliser les connaissances nécessaires à cette traduction.

4.1. Définitions

Une ontologie définit les concepts ($\mathbb{C}$), les propriétés ($\mathbb{P}$) et les individus ($\mathbb{I}$) d'un domaine, où $\mathbb{C}$, $\mathbb{P}$ et $\mathbb{I}$ sont trois ensembles disjoints. Les concepts et propriétés d'une ontologie sont définis à l'aide d'axiomes terminologiques ($\mathbb{A}$). Ces derniers sont des formules typiquement décrites en logique de description portant sur les concepts et propriétés. Les individus sont associés aux concepts et aux propriétés à l'aide de fonctions d'association respectivement notées $\mathfrak{I}^{\mathbb{C}}$ et $\mathfrak{I}^{\mathbb{P}}$.

Une structure d'ontologie $\mathbb{O}$ peut alors être représentée par un n-uplet $< \mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}} >$ consistant en :

– Un ensemble $\mathbb{C}$ de concepts;

– Un ensemble $\mathbb{P}$ de propriétés binaires;

– Un ensemble $\mathbb{A}$ d'axiomes terminologiques;

– Un ensemble $\mathbb{I}$ d'individus;

– Une fonction $\mathfrak{I}^{\mathbb{C}}$ associant à chaque concept un ensemble d'individus;

– Une fonction $\mathfrak{I}^{\mathbb{P}}$ associant à chaque propriété un ensemble de couples d'individus ou de couples formés d'un individu et d'une valeur.

4.2. Modélisation du comportement d'un environnement intelligent

La profusion des types de capteurs et des comportements qui peuvent leur être associés a rapidement mis en évidence le besoin de formalismes permettant de représenter un vocabulaire commun et possédant suffisamment de sémantique pour permettre une bonne interopérabilité. Dans cette optique, plusieurs travaux proposent l'usage d'une ontologie pour la description de réseaux de capteurs ou d'environnements intelligents. Parmi ces travaux, émergent, les projets *SOPRANO* (Klein *et al.*, 2007) et

SensorML[6]. Le cadre applicatif de *SOPRANO* consiste à fournir aux personnes âgées nécessitant une assistance, des services sensibles à leur environnement de vie et en mesure d'intégrer les connaissances sur leur situation actuelle. L'objectif de *SOPRANO* est la mise en place d'une infrastructure pour une variété de capteurs, actionneurs et services dans une maison intelligente. Cette infrastructure repose sur l'utilisation d'une ontologie qui représente le contexte courant et passé de la personne assistée et non le fonctionnement de son environnement. *SensorML* est une norme de l'*Open Geospatial Consortium*[7] qui fournit des modèles standard et un codage XML pour décrire des capteurs et des processus de mesure. *SensorML* offre un cadre dans lequel les caractéristiques géométriques et dynamiques des systèmes de détection et des capteurs peuvent être définies. De manière plus ciblée, les travaux de (Avancha *et al.*, 2004),(Wang, Turner, 2009), (Khattak *et al.*, 2011) et (Albreshne *et al.*, 2013) modélisent les réseaux de capteurs à l'aide d'ontologies afin de permettre l'interrogation sur les diverses connaissances représentées. Dans ce cas, la définition des ontologies repose en général sur une hiérarchisation profonde des concepts. à notre connaissance, il n'existe pas d'ontologie modélisant le comportement d'un environnement intelligent, *i.e.* les règles de comportement. Nous proposons donc une modélisation dont l'objectif est de définir des concepts, des propriétés et des règles de comportement suffisamment génériques pour représenter l'ensemble des types et des comportements des composants de n'importe quel environnement intelligent sans avoir à modifier sa structure et assez spécifiques pour permettre la distinction de ses différents composants.

L'ontologie que nous proposons permet de représenter le comportement d'un environnement intelligent, de guider l'analyse des spécifications utilisateur et de faciliter leur transformation en spécifications formelles. Elle a été conçue pour être indépendante du type d'environnement intelligent et des spécifications utilisateur. Pour cela, nous y distinguons deux niveaux de représentations : i) une ontologie de haut niveau conceptualisant le comportement générique d'un environnement intelligent (cf. section 4.2) ; ii) une instanciation de l'ontologie à partir de l'analyse des spécifications qui décrit le comportement spécifique aux besoins de l'utilisateur (cf. section 5).

4.2.1. Concepts de haut niveau

Lors de la conceptualisation d'une ontologie, une question fondamentale concerne la profondeur hiérarchique *i.e.* déterminer à quel point spécialiser les concepts de l'ontologie. Pour (Poli, 1996), il est important de se concentrer sur les concepts de haut niveau du domaine. Selon lui, une ontologie ne doit pas être vue comme un catalogue du monde ou une terminologie mais comme la structure au sein de laquelle catalogue et terminologie peuvent trouver une organisation appropriée. Au contraire, (Guarino, 1997) préconise qu'une ontologie soit de granularité profonde, et insiste sur l'importance d'inclure tous les concepts spécifiques du domaine. Cette dernière approche demande une analyse très fine du domaine. Dans le cas de la représentation

6. *Sensor Model Language Encoding Standard.*
7. http://www.opengeospatial.org/

d'environnements intelligents où le nombre de capteurs de types différents ne cesse d'augmenter cela pourrait constituer un travail continuel. Pour permettre que le modèle de représentation puisse s'adapter tant à l'environnement intelligent qu'aux besoins d'un utilisateur en évitant d'avoir à ajouter des concepts et relations, nous avons choisi d'abstraire ce qui distingue les composants les uns des autres, c'est à dire leurs types et leurs propriétés. Ainsi, nous avons choisi une hiérarchie peu profonde au lieu d'une hiérarchie exhaustive (cf. figure 2, schéma de gauche), mais qui permet quand même de distinguer les différents types de composants (cf. figure 2, schéma de droite). Ce niveau de représentation est suffisant pour l'identification des individus de chaque concept, car au sein d'une ontologie, les individus se distinguent en fonction de leurs valeurs de propriétés. Ainsi, de l'étude du domaine des environnements intelligents, il ressort que l'ensemble des capteurs et actionneurs sont principalement caractérisés par le type de phénomène qu'ils prennent en charge ainsi que les localisations de l'environnement qu'ils gèrent. De manière similaire, les phénomènes se distinguent par leur type et leur lieu d'apparition.

Nous distinguons donc deux sortes de concepts (cf. figure 5) : *Composant* et *Type*. Leurs sous-concepts sont définis comme suit :

1. Chaque sous-concept de *Composant* représente un ensemble d'individus partageant des propriétés qui les distinguent d'une part, des autres composants et d'autre part, les uns des autres. Les composants sont tous les objets/entités spécifiques au domaine (capteurs, actionneurs, localisations ...);

2. Chaque sous-concept de *Type* représente un ensemble de types spécifiques au domaine. Les individus *Type* étendent les types prédéfinis simples (entier, réel ...) par des types propres au domaine (type de phénomène, , type de localisation, marque, modèle ...), permettant de caractériser les composants de l'environnement intelligent.

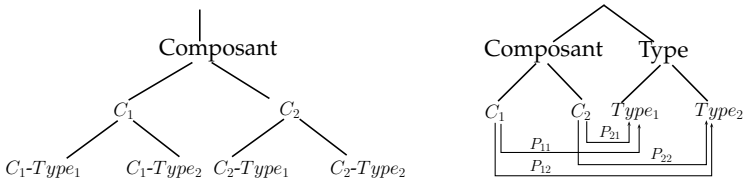

Figure 2. Hiérarchie profonde vs distinction entre composant et type

4.2.2. Propriétés variables et propriétés définissantes

Dans notre contexte, il est important d'une part de représenter l'aspect dynamique du comportement des composants de l'environnement de manière à permettre la traduction vers MAUDE et d'autre part de définir les conditions *nécessaires et suffisantes* à leur classification et identification pour guider l'analyse des textes. La définition des concepts doit alors s'accompagner de la description des propriétés représentant leurs interactions, l'évolution de leur état ainsi que les propriétés permettant de les identifier. Pour cela, nous distinguons en premier lieu deux types de propriétés : les relations, les attributs (cf. figure 3).

– Les relations décrivent une interaction entre deux composants. Par exemple, l'instance de propriété *Status-on*(*act-114*,*light-kitchen*) décrit l'action de mettre à l'état *on* le processus physique *light-kitchen* par l'actionneur *act-114*. Sous OWL, nous représentons une *propriété de type relation* comme une sous-propriété de la super-propriété *Relation*.

– Les attributs décrivent les caractéristiques des composants. Nous distinguons également deux types d'attributs (cf. figure 3) :

- attributs *variables* : leur valeur est amenée à évoluer au cours du temps, tel que l'état d'un processus physique (on, off ...). Il permettent de représenter l'évolution de l'état des composants. Par exemple, l'instance de propriété *Status*(*light-kitchen*,*on*) décrit l'état du processus physique *light-kitchen* à *on*. Pour représenter ce type d'attribut sous OWL, nous avons créé une propriété nommée *Attribut-variable* sous-propriété de *Attribut* (cf. figure 3).

- attributs *définissants* : leur valeur n'est pas amenée à évoluer, tel que le numéro de série d'un capteur ou son type. Ils permettent de distinguer de manières précises les composants les uns des autres. Par exemple, l'instance de propriété *Fixed-in*(*zigbee-A04*, *kitchen-ground-floor*) décrit l'emplacement physique du capteur *zigbee-A04* dans la localisation *kitchen-ground-floor*. Pour représenter ce type d'attribut sous OWL, nous avons créé une propriété nommée *Attribut-Définissant* sous-propriété de *Attribut* (cf. figure 3).

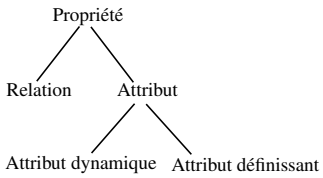

Figure 3. Hiérarchie des propriétés au sein de notre ontologie

4.2.3. Règles de comportement

Deux types de règles de comportement sont à distinguer : (1) les règles *génériques* qui décrivent le comportement générique que doit adopter l'environnement intelligent indépendamment de ses utilisateurs et (2) les règles *spécifiques* qui spécialisent le comportement de l'environnement intelligent désiré par l'utilisateur. À l'instar des concepts et propriétés propres au domaine des environnements intelligents, les règles *génériques* sont explicitées par l'expert du domaine. Quant aux règles *spécifiques*, elles seront créées par l'analyse des spécifications utilisateur.

Nous définissons une *règle de comportement* comme une formule, constituée d'un *antécédent* qui définit les conditions d'application de la règle et d'un *conséquent* qui définit le résultat de son application. On note *antécédent* $\rightarrow$ *conséquent*. Chacun des *antécédent* et *conséquent* est constitué de prédicats. Dans le paradigme des ontologies, un prédicat correspond à une instance de concept ou de propriété. Il a pour argument un individu, un littéral (valeur de type simple) ou une variable. Au sein d'une ontolo-

gie OWL, les règles dynamiques peuvent être représentées à l'aide de règles SWRL (Horrocks *et al*., 2004) formées de prédicats issus du vocabulaire conceptuel de l'ontologie. Néanmoins, le raisonnement sous OWL s'applique sans tenir compte des aspects potentiellement séquentiels ou concurrents des règles SWRL, ce qui peut mener à l'application simultanée de règles ne pouvant en réalité s'appliquer simultanément.

Nous avons donc choisi de représenter les règles de comportement au sein de l'ontologie de manière à pouvoir vérifier leur bonne formation et à pouvoir les traduire puis les exécuter sous Maude (cf. section 6). Pour les représenter, nous proposons de modéliser le concept *Patron-de-Règle* dont les instances, définies par le concepteur de l'ontologie, représentent les patrons de règles à rechercher dans les spécifications et le concept *Règle-Utilisateur* dont les instances représentent les règles de comportement spécifiées par l'utilisateur. Ces dernières seront créés automatiquement à partir de l'analyse des spécifications d'exigences. Ces deux concepts possèdent chacun les propriétés *Antécédent* et *Conséquent* qui ont comme valeur d'image un prédicat. Le type prédicat, n'étant pas défini sous OWL, nous avons défini le concept *Prédicat* sous-concept du concept *Type* (cf. figure 5). Il possède quatre propriétés *Nom-Prédicat* qui représentent un nom de concept ou de propriété, un *Type-Prédicat* (unaire ou binaire), un *Argument-1* et un *Argument-2* (individu, valeur de type OWL ou variable).

Dans une instance de *Patron-Règle*, les *prédicats à spécialiser* à partir de l'analyse des spécifications pour construire des instances de *Règle-utilisateur* sont ceux contenant au moins un argument à instancier (cf. figure 4) ou appartient au conséquent de la règle. Les prédicats définis dans le conséquent d'un patron de règles représentent des super-propriétés. Une super-propriété définit un type de propriété, par exemple dans notre modélisation la propriété *Actuate* définit le type des propriétés permettant d'actionner un appareil : *Turn-on*, *Turn-off*, *Increase*, *Decrease*. L'utilisation de super-propriétés permet de définir les types de propriétés à reconnaître dans les textes sans avoir à les énumérer. La figure 7 illustre un exemple d'instance du concept *Patron-Règle* (règle de gauche) contenant les arguments (sans « ? ») et prédicats (du conséquent) à spécialiser ainsi que l'instance de *Règle-Utilisateur* (règle de droite) contenant les prédicats spécialisés, créés automatiquement à partir de l'analyse des spécifications afin de spécialiser le patron de règle.

De manière similaire aux approches de (Gordon, Harel, 2009) ou (Kof, 2009) qui utilisent des modèles de représentation de type UML pour guider l'identification d'exigences, nous avons modélisé des patrons de règles d'exigences formés sur des prédicats ontologiques, définissant ainsi cinq types de règles à identifier à partir des textes :

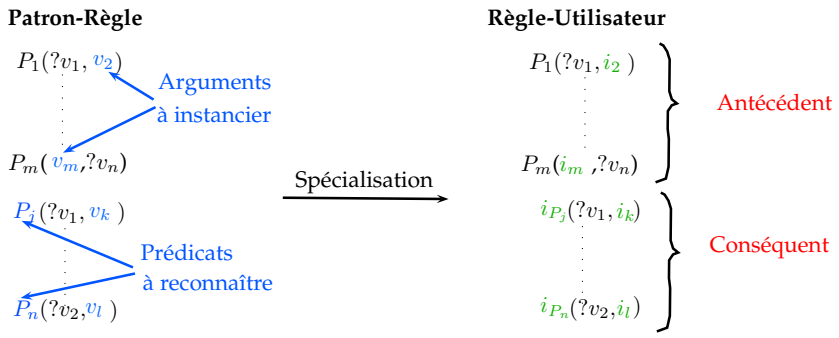

Figure 4. Spécialisation d'une instance de Patron-Règle

« PR1 »	« PR2 »	« PR3 »	« PR4 »	« PR5 »
Detected-in(t,lt)	Measured-in(m,lt)	Measured-in(m,lt)	Measured-in(m,lt)	
Controled-in(p,lt)	Controled-in(p,lt)	Controled-in(p,lt)	Controled-in(p,lt)	Controled-in(p,lt)
Has-type(?ph,t)	Has-type(?ph,m)	Has-type(?ph,m)	Has-type(?ph,m)	Has-type(?ph,p)
Occurred-in(?ph,?l)	Occurred-in(?ph,?l)	Occurred-in(?ph,?l)	Occurred-in(?ph,?l)	Occurred-in(?ph,?l)
Perceived-type(?s,t)	Perceived-type(?s,m)	Perceived-type(?s,m)	Perceived-type(?s,m)	Perceived-type(?s,m)
Sensing-zone(?s,?l)	Sensing-zone(?s,?l)	Sensing-zone(?s,?l)	Sensing-zone(?s,?l)	Sensing-zone(?s,?l)
Managed-type(?a,t)	Managed-type(?a,m)	Managed-type(?a,m)	Managed-type(?a,m)	Managed-type(?a,m)
Managed-zone(?a,?l)	Managed-zone(?a,?l)	Managed-zone(?a,?l)	Managed-zone(?a,?l)	Managed-zone(?a,?l)
Loc-type(?l,lt)	Loc-type(?l,lt)	Loc-type(?l,lt)	Loc-type(?l,lt)	Loc-type(?l,lt)
	Has-value(m,v)	greaterThan(m,v)	lessThan(m,v)	Has-state(p,st)
Actuate(?a,?p)	Actuate(?a,?p)	Actuate(?a,?p)	Actuate(?a,?p)	Actuate(?a,?p)

Le résultat de ces choix de conceptualisation permet de construire l'ontologie de haut niveau figurant en 1) sur la figure 5.

4.3. Ontologie du comportement d'un environnement intelligent

Nous allons maintenant décrire la partie de l'ontologie spécifique au domaine des environnements intelligents. Celle-ci, présentée en 2) sur la figure 5, comporte quatorze concepts : sept sous-concepts de *Composant*, et sept sous-concepts de *Type*. Les propriétés sont représentées par des flèches orientées qui lient les concepts de leur domaine aux concepts de leur image. Ces propriétés correspondent toutes à des *ObjectProperty* et sont au nombre de trente et une. Les flèches en pointillées représentent les relations de subsomption entre concepts. L'ontologie a été développée à l'aide de l'éditeur d'ontologie *Protégé*, sous la seconde version du langage d'ontologie web, OWL 2 [8].

Les concepts sous-concepts de *Composant*, *Sensor* et *Actuator*, représentent respectivement les capteurs et actionneurs installés dans l'environnement intelligent. Le concept *Phenomenon* décrit les phénomènes pouvant apparaître dans une des localisations de l'environnement intelligent, en lui associant un type *PhenomenonType* et une localisation *Location*. Les individus des concepts *Sensor*, *Actuator* et *Phenome-*

8. http://www.w3.org/TR/owl2-overview/

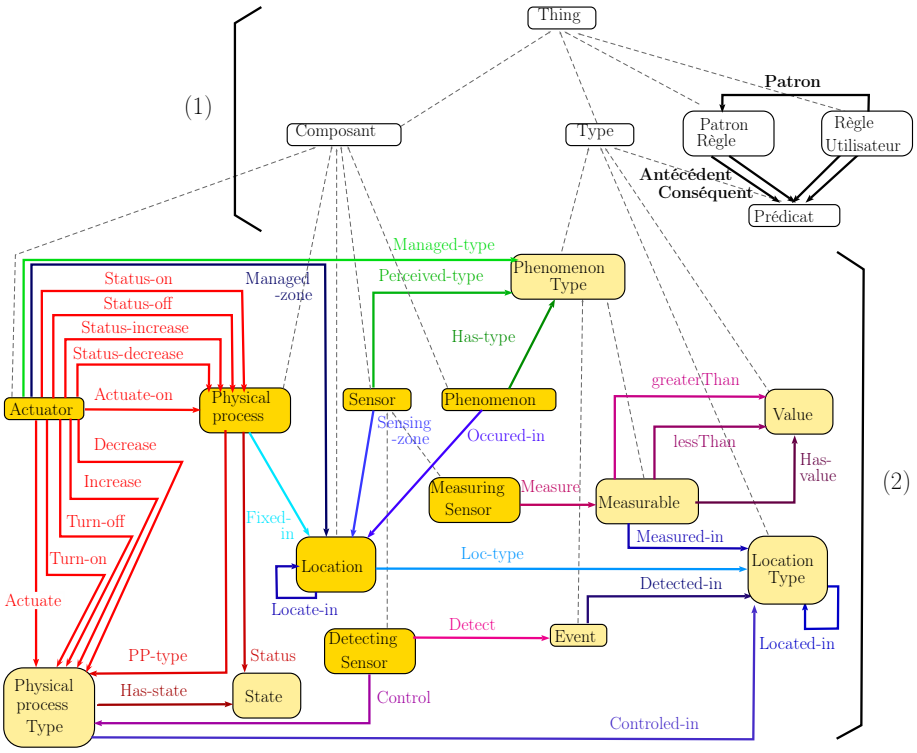

Figure 5. Ontologie de l'environnement intelligent

non sont donc tous identifiables par deux types de propriétés, l'une lui associant un type par une propriété *X-type* [9] et l'autre une localisation par une propriété *X-zone* ou *X-in* [10]. De manière analogue, une localisation est caractérisée par un type (*Location-Type*), qui a pour instance des types de localisation tels que *chambre*, *cuisine* ou *rez-de-chaussée* et peut elle-même être localisée dans une ou plusieurs localisations, ce qui permet d'exprimer par exemple le fait qu'une chambre est située au rez-de-chaussée.

4.3.1. Caractéristiques des propriétés de l'ontologie

Notre modélisation est fondée sur la définition de propriétés définissantes, de sorte à distinguer de manière univoque les individus de l'ontologie en fonction de leurs valeurs de propriétés. Chaque propriété est caractérisée par un seul concept de domaine et un seul concept d'image. Elles se déclinent donc au sein de l'ontologie en fonction des concepts qu'elles lient. Cela permet de distinguer des propriétés dont la sémantique est proche telles que *Fixed-in*, *Sensing-zone*, *Occured-in* ou *Controled-in* (cf.

9. avec X=perceived, managed et has respectivement.
10. X=Sensing, managed et occurred respectivement.

figure 5). Ce choix de modélisation permet de restreindre la sémantique véhiculée par chaque propriété et ainsi limiter les ambiguïtés d'interprétation lors de l'analyse des spécifications.

OWL offre la possibilité d'associer aux propriétés différentes caractéristiques. Les propriétés définissantes sont ainsi définies comme *fonctionnelles* ou *inverse fonctionnelles*. Les propriétés fonctionnelles (resp. *inverse fonctionnelles*) portent sur les individus ne pouvant posséder qu'une seule valeur d'image (resp. de domaine). Elles permettent d'inférer l'identité entre des individus, lorsque ceux-ci sont liés à une même valeur d'image au travers d'une même propriété fonctionnelle. Les propriétés transitives servent à inférer une nouvelle propriété entre le premier et le dernier individu à partir d'une suite d'individus qu'elles lient consécutivement. Elles permettent alors de limiter la quantité de connaissances à expliciter dans l'ontologie.

4.3.2. Définition de règles d'inférences

Dans certaines formulations, un terme peut connoter ou désigner une propriété qu'un individu doit posséder. La valeur de propriété de l'individu est alors caractérisée directement par le terme qui dénote cet individu dans le texte. La propriété n'est donc pas lexicalisée. C'est le cas de la propriété *Has-type* qui lie un individu du concept *Phenomenon* à un individu du concept *Phenomenon-Type*. *Has-type* n'est pas lexicalisée dans les textes. Elle est directement dénotée par des termes tels que *temperature*, *humidity*, *movement* qui font en même temps référence à un phénomène et à un type de phénomène. La valeur de la propriété *Has-type* doit donc être inférée à partir d'autres propriétés pouvant être extraites du texte tels que *Measured-in*, *Detected-in* et *Controled-in* liant chacune un individu de *Phenomenon-Type* à un individu de *Location-Type*. L'exemple ci-dessous (en a), montre comment au sein d'un patron de règle la valeur du type t de la propriété *Has-type* pour un phénomène $?p$ est induite à partir de sa valeur pour la propriété *Measured-in*. b) décrit un cas d'instanciation, *i.e.* une règle utilisateur où le type de phénomène reconnu est *temperature*.

a) ..., *Measured-in*(**t**,l) *Has-type*(?p,**t**) ... $\rightarrow$ ...

b) ... *Measured-in*(**temperature**, *kitchen*) *Has-type*($?p$, **temperature**) ... $\rightarrow$ ...

Enfin, une règle SWRL est définie pour chaque type de concept *Composant*. Elle permet de déduire les individus identiques à partir de leurs valeurs de propriétés définissantes, et ainsi lier les individus issus des textes aux individus pré-existant dans l'ontologie. La règle ci-dessous indique que deux capteurs $?s_1$ et $?s_2$ sont identiques s'ils perçoivent le même type de phénomène $?t$ et ceci dans la même zone de capture $?l$.

Perceived-type($?s_1$, $?t$) $\wedge$ *Perceived-type*($?s_2$, $?t$) $\wedge$ *Sensing-zone*($?s_1$, $?l$) $\wedge$ *Sensing-zone*($?s_2$, $?l$) $\rightarrow$ sameAs($?s_1$, $?s_2$)

subsectionConclusion L'ontologie que nous avons définie permet de représenter le comportement générique d'un environnement intelligent. Le second niveau de représentation correspond au peuplement de l'ontologie (cf. section 5). Afin d'être en

mesure d'extraire les informations permettant l'instanciation à partir de textes, nous avons associé une partie terminologique à l'ontologie, sous forme d'une termino-ontologie (Roche, 2008).

Une termino-ontologie est constituée d'un ensemble de termino-concepts. Un termino-concept est un terme désambiguïsé dont le sens est défini par son usage dans un corpus (Szulman, 2011). Il permet de faire le lien entre un concept de l'ontologie et les dénominations de ses individus dans les textes. De manière analogue, nous définissons la notion de termino-propriété comme un terme désambiguïsé dont le sens permet de faire le lien entre une propriété de l'ontologie et ses mentions dans les textes. Afin d'établir le lien entre terminologie et ontologie, nous utilisons le formalisme S*imple Knowledge Organization System* (SKOS [11]) qui permet d'associer à chaque instance de concept ou de propriété de l'ontologie une dénomination préférée ainsi qu'un ensemble de dénominations alternatives pouvant être rencontrées dans les textes. La terminologie est acquise automatiquement par amorçage (Sadoun *et al.*, 2013b).

5. Peuplement de l'ontologie à partir des spécifications analysées

Peupler une ontologie consiste à associer des individus aux concepts appropriés et définir les instances de propriété auxquelles participent les individus de l'ontologie. Dans notre contexte, l'objectif du peuplement de l'ontologie est de représenter le comportement du système à partir de l'analyse des spécifications utilisateur. Il s'agit de faire le lien entre les règles de comportement mentionnées dans les textes et leur conceptualisation dans l'ontologie.

En pratique, l'utilisateur doit décrire ses besoins en fonction de la configuration physique de son environnement intelligent, *i.e.* le type de ses composants, leur nombre ou leur emplacement et leurs types d'interactions. La description de ces informations doit se faire de manière préalable à l'analyse des besoins de l'utilisateur. Cette première instanciation de l'ontologie constitue un peuplement partiel (une initialisation). Celle-ci est peu sujette au changement car elle ne dépend pas des spécifications de l'utilisateur. Suite à l'initialisation de l'ontologie, toutes les instances qui représentent la configuration physique de l'environnement intelligent devront avoir été définies, à savoir les instances de capteurs, d'actionneurs, de processus physiques, de localisations et des différents types de phénomènes gérés par l'environnement. À titre d'exemple, lors de l'initialisation de l'ontologie, une instance est créée pour chaque capteur avec comme propriétés son *type sémantique* (*Detecting-Sensor*), sa *zone de détection* (*Sensing-zone*) et son *type de phénomène perçu* (*Perceived-type*) comme l'illustre la figure 6.

À partir de cette instanciation initiale, il est alors possible de guider l'identification des règles utilisateur décrites dans les spécifications.

11. http://www.w3.org/2004/02/skos/

– *Detecting-Sensor*(*zigbee-A04*) :
zigbee-A04 est une instance du concept *Detecting-Sensor*.

– *Sensing-zone*(*zigbee-A04*,*kitchen-ground-floor*) :
zigbee-A04 a comme zone de détection *kitchen-ground-floor*

– *Perceived-type*(*zigbee-A04*, *movement*) :
zigbee-A04 détecte des phénomènes de type *movement*.

Figure 6. Instanciation du capteur zigbee-A04

5.1. Identification des règles utilisateur

Afin de guider l'identification des règles utilisateur dans les textes, nous avons donc défini des patrons de règles au sein de l'ontologie représentés par les instances du concept *Patron-Règle* (cf. section 4.2.3). Chacune de ces instances correspond à un type de règle à identifier. La création d'une instance de *Règle-utilisateur* est guidée par l'exploitation d'un individu du concept *Patron-de-Règle* (cf. étape 2, algorithme 1). Plusieurs instances du concept *Règle-utilisateur* peuvent être associées à une instance du concept *Patron-de-Règle*. Chacune d'elles est une spécialisation d'une instance de *Patron-de-Règle*. La figure 7 illustre la spécialisation d'une instance du concept *Patron-de-Règle PR-3* (illustrée à gauche) à partir de l'analyse de la phrase numéro 1 *When I enter a room the door opens automatically* pour la création de l'instance *RU-1* du concept *Règle-Utilisateur* (illustrée à droite). Lors de l'analyse des spécifications, les prédicats à spécialiser (cf. section 4.2.3) définis par les propriétés *Antécédent* et *Conséquent* des instances de *Patron-de-Règle* sont recherchés dans les textes (cf. étape 1, algorithme 1). Les instances de propriété extraites de chaque phrase sont alors comparées aux prédicats à spécialiser (cf. étape 3, algorithme 1). Si tous les prédicats à spécialiser d'un patron ont été reconnus alors une règle utilisateur spécialisant ce patron est construite (cf. étape 4, algorithme 1). Avant la création il faut vérifier que l'instance de règle résultat n'est en contradiction avec aucune instance de règle de l'ontologie (cf. étape 5, algorithme 1). Si cette condition est vérifiée, alors une instance du concept *Règle-utilisateur* est créée (cf. étape 6, algorithme 1). À chaque instance de *Règle-Utilisateur* créée est associé le numéro de phrase dont elle est issue, à l'aide de la propriété *Num-phrase*, de sorte à préserver le lien entre les spécifications textuelles et les règles de comportement créées.

5.2. Identification des prédicats d'une règle utilisateur

Le peuplement de l'ontologie consiste à lier les mentions faisant référence aux composants du système et leurs types dans les textes aux individus correspondants de l'ontologie. Il n'est donc pas seulement question d'associer les individus aux concepts qui les dénotent mais aussi d'associer les individus décrits dans les spécifications aux individus préexistants dans l'ontologie lorsqu'ils sont identiques [12], *i.e.* qu'ils repré-

12. Deux individus sont identiques si leurs valeurs de propriétés définissantes sont identiques

Algorithme 1. Création des règles utilisateur guidée par les patrons

for *each sentence* S *in spécifications* **do**
// étape 1 - extraction des instances de propriétés extraites
$I_P \leftarrow getExtractedInstanceProperties$;
//Renvoie les patrons de règles à reconnaître
$\mathbb{I}_{PR} \leftarrow$ *getIndividuals*(*Patron-Règle*);
// étape 2 - guider l'identification des règles utilisateur
for *each* i_{PR} *in* $\mathbb{I}_{PR}$ **do**
$I_{P_{Found}} \leftarrow \emptyset$; $I_{P_{Missing}} \leftarrow \emptyset$;
//Renvoie les prédicats à spécialiser
$PR \leftarrow getPredicateToSpecialize(i_{PR})$;
// étape 3 - rechercher les prédicats à spécialiser
for *each* pr *in* PR **do**
//Renvoie les instances i_p qui spécialisent le prédicat pr
$i_P \leftarrow$ *getMatchingInstance*(pr, I_P);
if $i_P \neq \emptyset$ **then**
$I_{P_{Found}} \leftarrow I_{P_{Found}} \cup i_P$;
else
$I_{P_{Missing}} \leftarrow I_{P_{Missing}} \cup pr$;
if $I_{P_{Missing}} = \emptyset$ **then**
//étape 4 - construction une règle utilisateur
$i_{UR} \leftarrow buildUserRule(i_{PR}, I_{P_{Found}})$;
//Renvoie les règles en contradiction si elles existes
opposite-rules $\leftarrow$ *getOppositeRules*(i_{UR});
//étape 5 - vérification de la cohérence de la règle utilisateur
if *opposite-rules*$= \emptyset$ **then**
//étape 6 - création de l'instance de règle utilisateur identifiée
$createUserRequirement(i_{UR})$;
else
$print$(Les règles {*opposite-rules*} sont en contradiction avec i_{UR});
else
$print$(Les instances {$I_{P_{Missing}}$} sont manquantes !);

sentent une même entité. Ces liaisons sont permises par l'application d'inférences fondées sur l'exploitation des propriétés définissantes définies par notre modèle ontologique.

Les individus correspondent à des objets du monde et peuvent avoir plusieurs dénominations et une même dénomination peut correspondre à différents objets (cas d'ambiguïtés). Ils se distinguent alors par leurs caractéristiques, *i.e.* leurs types et propriétés. L'approche de peuplement que nous avons proposée (Sadoun *et al.*, 2013a) est fondée non pas sur l'identification des mentions d'individus dans les spécifications, qui peuvent être ambiguës ou même implicites, mais sur les valeurs de propriétés identifiées dans les textes, qui fournissent un contexte de désambiguïsation. En effet,

Instance *PR-3* du *Patron-de-Règle*
Detected-in(**t**,**lt**)
Controlled-in(**p**,**lt**)
Has-type(?ph,**t**)
Occurred-in(?ph,?l)
Perceived-type(?s,**t**)
Sensing-zone(?s,?l)
Managed-type(?a,**t**)
Managed-zone(?a,?l)
Loc-type(?l,**lt**)
→ **Actuate**(?a,**p**)

$\xrightarrow{\text{Spécialisation}}$

Instance *RU-1* de *Règle-Utilisateur*
Detected-in(**movement-in**,**room**)
Controlled-in(**door**,**room**)
Has-type(?ph,**movement-in**)
Occurred-in(?ph,?l)
Perceived-type(?s,**movement-in**)
Sensing-zone(?s,?l)
Managed-type(?a,**movement-in**)
Managed-zone(?a,?l)
Loc-type(?l,**room**)
→ **Turn-on**(?a,**door**)

Figure 7. Exemple de patron de règle et règle utilisateur

le type de concept d'un individu issu d'une formulation ambiguë peut être inféré dès lors qu'il est reconnu comme domaine ou image d'une instance de propriété (cf. section 4.3.1). De plus, l'identité d'un individu présent de manière implicite peut être déduite par la reconnaissance de ses propriétés définissantes. Par exemple, selon les choix de modélisation [13] une *voiture* pourra être reconnue par son *numéro d'immatriculation* ou *sa marque et son modèle*, de même, une *personne* pourra être reconnue par son *numéro de sécurité sociale* ou ses *nom, prénom et date de naissance*. Guider le peuplement par l'identification d'instances de propriété nous permet de faire appel aux mécanismes d'inférence de l'ontologie pour classer et identifier de manière univoque les individus participant aux instances de propriété extraites des spécifications.

Le résultats de cette identification de prédicats constitue le point d'entrée du processus d'identification de règles utilisateur. Dans notre approche, les prédicats sont extraits de manière automatique. Néanmoins, ces derniers pourraient aussi bien être donnés manuellement ou en utilisant une autre approche d'extraction que celle que nous avons proposé dans (Sadoun *et al.*, 2013a).

5.3. Classification et identification des individus

Le raisonnement sur l'ensemble des individus permet de les classer et de les identifier de manière unique. Deux types de raisonnement interviennent dans la classification d'un individu : en fonction du domaine ou de l'image des propriétés auxquels il est associé ou à partir des conditions *nécessaires et suffisantes* de chaque concept (cf. section 4.2.2). L'identification des individus se fait par raisonnement sur leur propriétés définissantes *i.e.* les propriétés fonctionnelles ou inverses fonctionnelles (cf. section 4.3.1) pour les individus pouvant être distingués par une seule propriété ou à

13. Le lecteur prendra note que l'ensemble des exemples décrits dans ce manuscrit sont supposés correspondre à un choix de modélisation particulier.

l'aide de règles SWRL (cf. section 4.3.2) lorsque les individus nécessitent plus d'une propriété pour être distingués des autres.

Lors de sa création, chaque individu est décrit par trois valeurs de propriétés correspondant à son numéro de phrase, son numéro de nœud et à sa formulation dans les spécifications. Ces connaissances permettent de maintenir le lien entre les instances de l'ontologie et leurs formulations dans les textes.

5.4. Vérification de la cohérence de l'ontologie instanciée

Les instances nouvellement créées sont potentiellement introductrices d'incohérences. La cohérence de l'ontologie résultat doit donc être vérifiée. Cette vérification est faite à chaque nouvelle instanciation, *i.e.* pour chaque phrase analysée, de sorte à faire ressortir les éventuelles erreurs au plus tôt. Dans le cas ou les nouvelles instances introduisent des incohérences, l'instanciation en cause est annulée et le processus de peuplement reprend à partir de l'état précédent de l'ontologie (où celle-ci est cohérente). Les incohérences sont souvent le résultat de deux instances dont les informations sont en contradiction. Les instances responsables sont alors exhibées par le raisonneur *Pellet* (Sirin *et al.*, 2007). Le lien maintenu au sein de l'ontologie entre les instances et leurs mentions dans le texte permet alors de pointer les erreurs de spécifications à corriger.

La validation complète des spécifications d'exigences nécessite la simulation du comportement de l'environnement intelligent qu'elles décrivent, afin de vérifier que tous les états désirés peuvent être atteints et que l'ensemble des états non désirés ne l'est pas. Le langage de spécifications formelles *Maude* permet ce type de validation grâce à son model-checker (Eker *et al.*, 2004).

6. De l'ontologie aux spécifications formelles Maude

Maude [14] est fondé sur la logique de réécriture et permet de décrire les changements d'états d'un système à l'aide d'une théorie équationnelle et d'un ensemble de règles de réécriture. Maude offre une syntaxe orienté-objet adaptée à la définition de systèmes concurrents. Le mécanisme de réécriture permet d'animer une spécification et de vérifier certaines propriétés comme par exemple, l'atteignabilité ou la non-atteignabilité de certains états.

En Maude, l'espace d'états d'un système est représenté par une spécification équationnelle $(\Sigma, \mathcal{E})$, où Σ est une signature qui défini des sortes (des types) des objets, constantes et variables manipulés par Maude et des opérateurs qui opèrent sur les données manipulées. $\mathcal{E}$ est l'ensemble des axiomes équationnels constitués des termes de la signature.

14. http://maude.cs.uiuc.edu/

La dynamique du système est décrite à l'aide de règles de réécriture. Ces règles décrivent les transitions simultanées possibles dans le système. L'application de chaque règle engendre une évolution de l'état du système. Les règles de réécriture sont de la forme $R : t \rightarrow t^{'}$ avec t et $t^{'}$ des termes de Σ. Ainsi le terme t se réécrit en terme $t^{'}$.

Le processus de transformation de l'ontologie OWL en spécification formelle Maude est résumé par la figure 8. Il prend en entrée les résultats de requêtes sur l'ontologie (spécifications OWL). Ces requêtes correspondent à l'application de fonctions de manipulation de l'ontologie implantées à l'aide des API Java *OWL-API* et *Jess*. Ces fonctions ont pour objectif de sélectionner les éléments ontologiques nécessaires à la représentation du comportement de l'environnement intelligent. La traduction du modèle ontologique en un modèle Maude est suivie par l'application de fonctions dites de *pretty-printing* que nous avons écrites qui génèrent du code Maude à partir des éléments du modèle Maude obtenu. L'application de l'ensemble des fonctions de *pretty-printing* engendre le document de spécifications Maude. Ce mode de transformation a l'avantage de ne pas se conformer à la syntaxe des langages qui peut être amenée à évoluer, ou même se décliner sous plusieurs versions. Cette transformation est détaillée dans (Sadoun *et al.*, 2014).

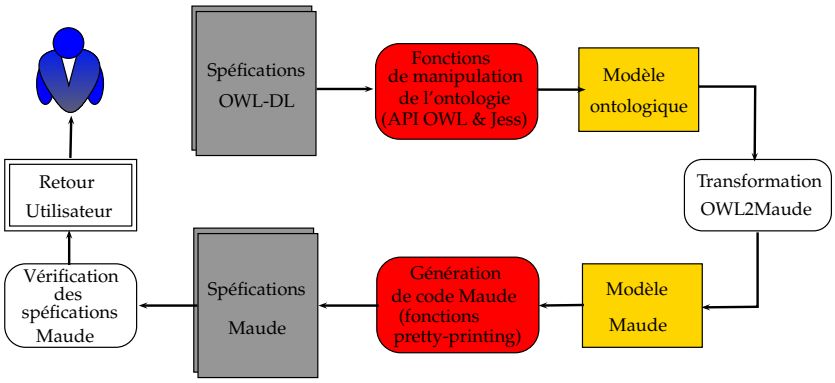

Figure 8. Transformation et vérification de l'ontologie OWL sous Maude

La figure 9 illustre la *règle de réécriture RR-1* créée à partir de la règle utilisateur *RU-1* (cf. figure 7), issue de la phrase numéro 1 *When I enter a room the door opens automatically.*

Sous Maude, la vérification porte principalement sur la cohérence du modèle (cf. section 7.4.5) et la conformité des règles utilisateur (cf. section 7.4.6) car celle-ci nécessite l'exploration par model-checking des états possibles du système. Cela consiste à définir les états du système que l'on peut souhaiter atteindre ou ne jamais atteindre. Ces états sont données par un expert qui connaît les contraintes du domaine à respecter. Tout au long de la traduction de OWL à Maude, Le lien entre les instances OWL et les objets Maude est préservé en nommant chaque règle de réécriture Maude à partir du numéro de phrase dont elle est issue et chaque objet Maude à partir du numéro d'instance dont il est issu.

rl [RR-1] : < **door** : **Physical-process-Type** | Controlled-in : **room** >
< I1-445-8 : **Location** | Loc-type : **room** >
< I1-326-6 : **Phenomenon** | Has-type : movement-in, Occurred-in : I1-445-8 >
< I1-280-7 : **Sensor** | Perceived-type : movement-in, Sensing-zone : I1-445-8 >
< **room** : **Location-Type** | >
< smoke : **Event** | Detected-in : **room** >
< I1-8-2 : **Actuator** | Managed-type : movement-in, Managed-zone : I1-445-8 >
$\longrightarrow$
< **door** : **Physical-process-Type** | Controlled-in : **room** >
< I1-445-8 : **Location** | Loc-type : **room** >
< I1-326-6 : **Phenomenon** | Has-type : movement-in, Occurred-in : I1-445-8 >
< I1-280-7 : **Sensor** | Perceived-type : movement-in, Sensing-zone : I1-445-8 >
< room : **Location-Type** | >
< smoke : **Event** | Detected-in : **room** >
< I1-8-2 : **Actuator** | Managed-type : movement-in, Managed-zone : I1-445-8,
Turn-on : **door** > .

Figure 9. Règle de réécriture Maude traduite de la règle utilisateur RU-1

7. Application et résultats

Dans cette section, nous présentons les résultats obtenus à partir d'un corpus de spécifications d'exigences recueilli via une plate-forme d'acquisition web.

7.1. Corpus de test

Afin de recueillir des spécifications d'exigences utilisateur portant sur la configuration du comportement d'un environnement intelligent, nous avons mis en place une plate-forme d'acquisition. Cette plate-forme est accessible sur le web [15]. Une fois l'utilisateur connecté, une interface de rédaction des spécifications lui est proposée. Cette interface décrit de manière graphique et textuelle la configuration physique de l'environnement à configurer ainsi que les différents dispositifs qui y sont intégrés. Les spécifications acquises sont constituées d'une centaine de phrases (2 171 mots) décrites par une vingtaine de personnes. Celles-ci ont été considérées comme une seule spécification et annotées manuellement en instances de propriété et règles de comportement afin de permettre l'évaluation de notre approche.

7.2. Identification des règles utilisateur

L'identification des règles utilisateur dépend principalement des instances de propriétés extraites des spécifications. L'extraction automatique de ces instances s'est faite avec une précision élevé d'environ 85 % et un rappel moins élevé d'environ 47

15. http://perso.limsi.fr/sadoun/Application/fr/SmartHome.php

% (voir (Sadoun, 2014) pour plus de détails). En tout, sur 345 prédicats présents, 162 ont été reconnus correctement, 30 ont été reconnus incorrectement et 183 n'ont pas été reconnus. À partir des prédicats extraits, 28 règles de comportement ont été complètement identifiées sur un total de 62 règles annotées (cf. tableau 1). Les 34 règles restantes ont été identifiées, mais de manière incomplète, car certains de leurs prédicats n'ont pas été reconnus (cf. section 7.4.4). Les prédicats non reconnus ont pour cause un terme absent de la termino-ontologie ou une formulation syntaxique non connue de l'extracteur automatique. Parmi les 28 règles construites, les règles comportant des erreurs sont au nombre de 5. Elles ont pour cause l'un des deux cas suivants :

1. une instance de propriété incorrecte pour 2 règles. Par exemple à partir de la phrase ... *detecting smoke in the kitchen* ... est extraite l'instance de propriété *Detected-in*(*movement-in*,*kitchen*) au lieu de *Detected-in*(*smoke*,*kitchen*).

2. une analyse syntaxique incorrecte pour 3 règles qui a pour résultat la création de prédicats supplémentaires. Ce type d'erreur est relevé sous *Maude* (cf. section 7.4.2).

Sur les 5 règles comportant des erreurs, 3 règles ont pu être automatiquement identifiées comme erronées (cf. section 7.4.2). Les autres ont nécessité un contrôle humain pour être identifiées comme non conformes. Ce résultat montre l'importance de privilégier la précision par rapport au rappel des connaissances extraites des textes.

Par ailleurs, on dénombre 5 règles utilisateur bien formées auxquelles cependant il manque l'un des prédicats du conséquent *i.e.* une des actions à effectuer n'a pas été identifiée. Par exemple, à partir de la phrase 59 *Open the windows and close the window shades when it's too warm inside a room.* le prédicat du conséquent sur *Open the windows* a été extrait en revanche le prédicat portant sur *close the window* n'a pas été extrait.

Tableau 1. Résultats de l'identification des règles utilisateur

Type règle	**Nb**	**Commentaire**
Référence	62	annotées
complète	28/62	prédicats tous identifiés
incomplète	34/62	prédicats manquants (non identifiés)
Correct	18/28	tous les prédicats présents et corrects
Incorrect	5/28	2 prédicats incorrects, 3 prédicats supplémentaires
Conséquent manquant	5/28	un des prédicats du conséquent manque (non identifié)

Si seule la moitié ($\simeq$ 47 %) des instances de propriété attendues sont extraites, l'autre moitié n'est pas oubliée. Tout au long du processus d'extraction d'instances et d'identification des règles, différents types d'information sur les spécifications sont collectés. Ces informations pointent les instances de propriété manquantes dans les spécifications ainsi que les phrases qui contiennent potentiellement une règle utilisateur. Ces retours servent à l'amélioration des spécifications utilisateur et du système d'extraction de manière itérative.

7.3. Résolution des ambiguïtés et identification de l'implicite

Le tableau 2 présente les résultats de la désambiguïsation des termes communs aux ensembles de termes *Physical-process* et *Actuate* tels que *light* et les ensembles de termes *State* et *Actuate* tels que *open* ou *close*. Dans le premier cas d'ambiguïté, tous les termes ont été désambiguïsés correctement. Dans le second cas, plus de 90 % des termes ont été désambiguïsés correctement. Ces résultats confirment la pertinence d'avoir recours aux contextes syntaxico-sémantiques des termes pour identifier les entités auxquelles ils font référence.

Tableau 2. Résultats de la résolution des termes ambigus

Termes ambigus	**Correct**	**incorrect**
Physical-process/Actuate	30	0
State/Actuate	73	6

L'originalité de cette approche est qu'outre l'extraction d'instances de propriété à partir des spécifications utilisateur, elle permet l'identification d'instances de propriété et de concept implicites. Le tableau 3 présente les instances de propriétés inférées à partir des instances de propriété extraites ainsi que les individus créés à partir des instances de propriété inférées.

Tableau 3. Instances de propriété et de concept implicites créées

Instance de propriété	**Nombre**	**Individu**	**Nombre**
Has-type(ph,typ)	28	Phenomenon	28
Occurred-in(ph,loc)	28		
Perceived-type(s,typ)	28	Sensor	28
Sensing-zone(s,loc)	28		
Managed-type(a,typ)	28	Actuator	28
Managed-zone(a,loc)	28		

7.4. Diagnostics d'erreurs retournées par le système

Les erreurs identifiées dans les spécifications peuvent être dues à l'utilisateur ou au système d'analyse des spécifications. Les diagnostics d'erreurs servent à l'utilisateur pour la correction de ses spécifications ou à l'amélioration du système d'analyse.

7.4.1. Gestion de la cohérence des règles de comportement

Deux règles sont en contradiction lorsqu'elles partagent le même antécédent et qu'elles concluent sur deux conséquents opposés. Par exemple, la règle *R-84* issue de la phrase numéro 84 [16] a été identifiée comme étant en contradiction avec la règle *R-1*

16. La numérotation des phrases est calculée en fonction de la segmentation des spécifications en phrases qu'effectue l'analyseur syntaxique.

issue de la phrase numéro 1 (cf. figure 10). Les individus en gras correspondent aux connaissances similaires issues des spécifications et les propriétés surlignées correspondent aux propriétés en contradiction. Le nom de la variable *?I84-9* qui désigne un individu du concept *Actuator* indique qu'elle est issue de la phrase numéro 84 et du nœud numéro 9 de l'arbre de dépendances syntaxiques de la phrase. Le nom de la variable ?I84-227 qui désigne un individu du concept *Sensor* indique qu'elle est issue de la phrase numéro 84 et du numéro aléatoire 227 compris entre 100 et 500, car la référence à sa propriété *Perceived-type* est implicite dans le texte. Au total 4 règles sur 28 ont été reconnues contradictoires. Ces règles ainsi que les phrases qui leur sont associées, sont stockées pour être soumises à l'utilisateur.

L'application simultanée de certaines règles possédant des antécédents différents peut aussi mener à des conséquents opposés. Ce cas nécessite l'exploration du modèle sous Maude (cf. section 7.4.5) afin de vérifier le résultat d'exécution des règles de comportement.

R-84 : Each time someone is detected in a room shut its doors.
Detected-in(**movement-in**,**room**) *Controled-in*(**door**,**room**)
Has-type(?I84-114,**movement-in**) *Occurred-in*(?I84-114,?I84-271)
Perceived-type(?I84-227,**movement-in**) *Sensing-zone*(?I84-227,?I84-271)
Managed-type(?I84-9,**movement-in**) *Managed-zone*(?I84-9,?I84-271)
Loc-type(?I84-271,**room**) ⇒ Turn-off(?I84-9,**door**)
R-1 : When I enter a room the door opens automatically.
Detected-in(**movement-in**,**room**) *Controled-in*(**door**,**room**)
Has-type(?I1-326,**movement-in**) *Occurred-in*(?I1-326,?I1-445)
Perceived-type(?I1-280,**movement-in**) *Sensing-zone*(?I1-280,?I1-445)
Managed-type(?I1-8,**movement-in**) *Managed-zone*(?I1-8,?I1-445)
Loc-type(?I1-445,**room**) ⇒ Turn-on(?I1-8,**door**)

Figure 10. Règles utilisateur extraites et reconnues contradictoires

7.4.2. Gestion des règles mal formées

La *phrase numéro 20* correspond à une combinaison de phrases segmentées de manière incorrecte par l'analyseur syntaxique.

Phrase numéro 20 – *As I dislike warmth I would dream of a blind that would be lowered everytime luminosity is too bright meaning a temperature up to 20 C. I do not want my cats to fall from upstairs so windows at this floor must be closed on two conditions : condition one is that there is no human in the room with the open window, condition two is that there is no cat on the windows still !! The only exception of course is if there is smoke in the house : in this case, and in this case only, open all the windows.*

Cette erreur de segmentation de phrase a eu comme répercussion une erreur sur la règle qui en a été extraite. La règle extraite porte uniquement sur la dernière phrase *The only exception ... open all the windows.* mais elle contient dans son conséquent en plus du prédicat correct *Turn-on*(?I20-103,window) le prédicat *Turn-off*(?I20-45,

window) qui est incorrect car il est issu de la phrase *I do not want ... must be closed on two conditions*. Lors de l'identification de la règle utilisateur, la variable *?I20-45* est identifiée comme incorrecte. Cela à l'aide de requêtes simples sur les prédicats de la règle utilisateur, car la variable *?I20-45* apparaît dans le conséquent de la règle sans être définie dans son antécédent. Ce type de vérification nous a permis d'identifier de manière automatique 3 des 10 règles mal formées.

7.4.3. Gestion des règles non applicables

Les règles non applicables sont celles dont un prédicat ne peut être satisfait, *i.e.* instancié. Ce cas correspond au fait que la configuration physique de l'environnement intelligent n'est pas en adéquation avec les besoins utilisateur. Par exemple, à partir de la phrase numéro 20, une règle comportant les prédicats : *Perceived-type*(?s,smoke), *Managed-type*(?a,smoke) et *Actuate*(?a,window) a été reconnue. Cette règle a été identifiée comme non applicable car la configuration physique de l'environnement intelligent représentée au sein de l'ontologie ne définit aucun capteur ($?s$) qui perçoivent le type *smoke* , aucun actionneur ($?a$) gérant ce même type et aucun actionneur ($?a$) pouvant effectuer une action sur un processus physique de type *window*. À partir de la configuration physique de l'environnement modélisée par l'ontologie, 20 règles utilisateur ont été identifiées comme non applicables. Dans ce cas, les instances en cause, c'est-à-dire manquantes, sont transmises à l'utilisateur qui pourra choisir entre abandonner sa règle ou veiller à ce que la configuration physique de l'environnement intelligent évolue. Les prédicats manquants [17] dans l'ontologie ont été ajoutés afin de permettre l'exécution des règles de comportement sous *Maude*.

7.4.4. Gestion des règles incomplètes

Les règles n'ayant pu être complètement identifiées sont stockées de sorte à permettre d'identifier les causes de leur non-identification. Pour cela, une heuristique simple est appliquée. Elle a pour objet de renvoyer le maximum de phrases pouvant contenir une règle utilisateur, *i.e.* les phrases desquelles au moins une instance de propriété a été extraite sans pour autant avoir entraîné l'identification d'une règle utilisateur. Ces phrases sont considérées comme contenant une règle potentiellement incomplète. En tout, 54 phrases ont été stockées comme potentiellement porteuses de règles utilisateur. Celles-ci contiennent bien les 34 règles n'ayant pas été identifiées (cf. tableau 1). La phrase numéro 2 *If it is nighttime the light bulbs of the room switch on.* est un exemple de phrase stockée contenant une règle potentielle. De cette phrase, seuls les deux prédicats : *Turn-on*(I2-11,light) et *Controled-in*(light,room) ont été identifiés, ce qui n'a pas suffi à l'identification de la règle spécifiée. Ce retour permet donc d'identifier des spécifications incomplètes ou des éléments pouvant compléter la modélisation du comportement du système. Par exemple, dans notre modélisation, le terme *nighttime* au même titre que les termes *Sunday* ou *Summer* n'est pas

17. Qui décrivent la configuration physique de l'environnement intelligent.

considéré comme un phénomène, caractérisé par un type et une localisation. Il dénote un cadre temporel dont la prise en compte figure dans nos perspectives.

7.4.5. Vérification de la cohérence du modèle

Sous Maude, l'exploration du modèle représenté est possible grâce à la fonction *search* qui permet à partir d'une configuration initiale d'inspecter les différents états du modèle en fonction de l'application des règles de réécriture. Dans notre contexte, la configuration initiale que nous appelons $init$, contient l'ensemble des individus de l'ontologie issus des spécifications utilisateur ou décrivant la configuration physique de l'environnement intelligent.

Le temps d'exécution des règles étant potentiellement infini, la vérification de l'état du système est menée par palier. La commande *search* permet de limiter le nombre successif de règles de réécriture appliquées à partir d'une configuration initiale. Les deux requêtes ci-dessous sont un exemple d'exploration du modèle par pallier. A partir de la configuration initiale *init* chacune des requêtes exécute un certain nombre de réécritures à la recherche d'une configuration C qui serait accompagnée des deux messages opposés *Status-off*(a-1,p-1) et *Status-on*(a-1,p-1) entre un actionneur *a-1* et un processus physique *p-1*. Ces deux messages provoquent chacun un changement d'état opposé de *p-1*, respectivement « on » ou « off ». La requête 1 qui applique 3 réécritures successives permet de trouver une solution *i.e.* un état du système conforme à la requête. La requête 2 qui applique 4 réécritures permet de trouver 39 solutions. Dans l'ensemble des solutions, les objets en cause sont l'actionneur *a-door-room* et le processus physique *door-room*. Les trois règles *R-1*, *R-88* et *R-103* sont en cause. *R-1* et *R-103* déclenchent un *Turn-on* respectivement lorsqu'un mouvement est détecté (*Detected-in*(*movement-in*,*room*)) ou qu'une fenêtre est contrôlée ouverte (*Controled-in*(*window*,*room*) et Has-state(*door*,*on*)). Quant à *R-88*, elle déclenche un *Turn-off* lorsque de la fumée est détectée (*Detected-in*(*smoke*,*room*)). Ces trois règles et leur phrases associées sont donc soumises à l'utilisateur qui jugera des précisions à y apporter. La figure 11 présente les deux règles *R-1* et *R-88* dont l'application simultanée peut mener à une incohérence.

Requête 1 – *search [,**3**] in SmartHome :*
init ⇒+ C:Configuration Status-off(a-11,p-11) Status-on(a-1,p-1) .
1 solution 1
C:Configuration ; a-1:Oid → a-door-room ; p-1:Oid → door-room

Requête 2 – *search [,**4**] in SmartHome :*
init ⇒+ C:Configuration Status-off(a-1,p-1) Status-on(a-1,p-1) .
39 solutions
C:Configuration ; a-1:Oid → a-door-room ; p-1:Oid → door-room

7.4.6. Vérification de la conformité des règles de comportement

Il s'agit de vérifier que le comportement du système est conforme aux exigences formulées par l'utilisateur, ou simplement conforme aux contraintes du domaine des environnements intelligents. Une vérification importante concerne la complétude du

Règle de réécriture issue de la phrase numéro 1
rl [R-1] :
< **door** : **Physical-process-Type** | Controled-in : **room** >
< I1-445-8 : **Location** | Loc-type : **room** >
< I1-326-6 : **Phenomenon** | Has-type : movement-in, Occurred-in : I1-445-8 >
< I1-280-7 : **Sensor** | Perceived-type : movement-in, Sensing-zone : I1-445-8 >
< **room** : **Location-Type** | >
< smoke : **Event** | Detected-in : **room** >
< I1-8-2 : **Actuator** | Managed-type : movement-in, Managed-zone : I1-445-8 >
→
< **door** : **Physical-process-Type** | Controled-in : **room** >
< I1-445-8 : **Location** | Loc-type : **room** >
< I1-326-6 : **Phenomenon** | Has-type : movement-in, Occurred-in : I1-445-8 >
< I1-280-7 : **Sensor** | Perceived-type : movement-in, Sensing-zone : I1-445-8 >
< room : **Location-Type** | >
< smoke : **Event** | Detected-in : **room** >
< I1-8-2 : **Actuator** | Managed-type : movement-in, Managed-zone : I1-445-8,
Turn-on : **door** > .

Règle de réécriture issue de la phrase numéro 88
rl [R-88] :
< **door** : **Physical-process-Type** | Controled-in : **room** >
< I88-148-9 : **Location** | Loc-type : **room** >
< I88-367-5 : **Phenomenon** | Has-type : smoke, Occurred-in : I88-148-9 >
< I88-51-9 : **Sensor** | Perceived-type : smoke, Sensing-zone : I88-148-9 >
< **room** : **Location-Type** | >
< smoke : **Event** | Detected-in : **room** >
< I88-18-9 : **Actuator** | Managed-type : smoke, Managed-zone : I88-148-9 >
→
< **door** : **Physical-process-Type** | Controled-in : **room** >
< I88-148-9 : **Location** | Loc-type : **room** >
< I88-367-5 : **Phenomenon** | Has-type : smoke, Occurred-in : I88-148-9 >
< I88-51-9 : **Sensor** | Perceived-type : smoke, Sensing-zone : I88-148-9 >
< **room** : **Location-Type** | >
< smoke : **Event** | Detected-in : **room** >
< I88-18-9 : **Actuator** | Managed-type : smoke, Managed-zone : I88-148-9,
Turn-off : **door** > .

Figure 11. Règles de réécriture R-1 et R-88

système par l'analyse de l'atteignabilité de certains états. Par exemple, dans le contexte d'un environnement intelligent, il est nécessaire de vérifier que chaque processus physique peut atteindre l'état *éteint* (off) et *allumé* (on) dans au moins un état du système. Par exemple, les requêtes 3 et 4 explorent les différents états du système à la recherche respectivement d'un état où le processus physique *light-bathroom* a comme valeur d'attribut *Status* : « on » ou « off ». La requête 3 renvoie deux solutions à partir de deux applications (n = 2) successives des règles de réécriture. En revanche, la requête 4 n'a pas donné de résultat après 5 itérations. On en déduit que la modélisation est probablement incomplète et qu'une règle permettant de passer de l'état du processus physique *light-bathroom* à « off » doit être définie. Un passage en revue des règles de réécriture confirme l'absence d'une telle règle. Le processus physique *light-bathroom* peut donc être allumé mais il ne peut être éteint dans la configuration actuelle. Les spécifications utilisateur doivent alors être augmentées d'une règle décrivant les conditions d'extinction de ce dernier.

Requête 3 – *search [,**n**] in OWL :*
init =>+ C:Configuration
< light-bathroom : Physical-process | Status : on > .

Requête 4 – *search [,**n**] in OWL :*
init =>+ C:Configuration
< light-bathroom : Physical-process | Status : off > .

Une fois toutes les vérifications nécessaires appliquées aux spécifications Maude, l'environnement intelligent peut, s'il est validé comme cohérent et conforme, être déployé. Suite à une modification des spécifications utilisateur, toute la configuration décrite doit être réévaluée.

8. Conclusion

Les ontologies permettent de définir les concepts et propriétés d'un domaine de manière formelle. Cette définition, lorsqu'elle est précise, permet l'application de mécanismes d'inférence dont peut résulter la déduction de nouvelles connaissances et de mécanismes de vérification permettant de valider ou d'invalider la cohérence des connaissances modélisées par l'ontologie. Néanmoins, jusqu'ici l'emploi de telles ontologies aussi bien pour l'analyse de textes en LN que pour la vérification de spécifications a été peu exploré.

Ce travail a été réalisé avec un objectif précis : répondre à la question : comment rendre un environnement intelligent aisément adaptable ? En effet, le moyen le plus simple pour un utilisateur d'exprimer ses besoins est de les spécifier en langage naturel. En partant de là, nous avons ainsi montré dans un premier temps qu'il était possible de produire, à partir des spécifications LN, une représentation ontologique décrivant les règles de comportement de l'environnement. Nous avons également montré qu'à partir de cette ontologie on était en mesure de produire un retour à l'utilisateur sur la qualité des spécifications d'exigences qu'il aura rédigées, à savoir si celle-ci sont complètes, cohérentes et conformes aux contraintes du domaine.

Dans ce travail, nous avons plus spécifiquement exploité les possibilités de représentation et de vérification offertes par OWL pour l'analyse et la formalisation de spécifications en LN, ce qui nous a permis de proposer un processus complet de traitement automatique de spécifications utilisateur avec une application aux environnements intelligents. Le fait de guider l'analyse des textes par les connaissances représentées dans l'ontologie permet d'identifier de manière fiable des instances de propriétés en se fondant sur des critères lexicaux, syntaxiques et sémantiques. La reconnaissance d'instances de propriétés entre entités permet dans un deuxième temps d'inférer les instances de concepts qui leur sont associées et de peupler l'ontologie. De plus, nous avons montré que l'ontologie instanciée peut alors être transformée de manière automatique en spécifications formelles Maude afin d'étendre la vérification au comportement de l'ensemble du modèle.

L'approche que nous proposons est suffisamment générique pour que l'on puisse embrasser l'idée de l'adapter à différents domaines d'application, dès lors que le comportement du système à modéliser peut s'exprimer sous forme de règles génériques. Les connaissances du domaine sont représentées de manière à être stables même si l'environnement d'application change : ce sont les valeurs associées aux instances, et donc la terminologie, qui doivent être adaptées, et non la structure conceptuelle. Ainsi, en perspective à ce travail, nous prévoyons de démontrer le caractère générique de notre approche de modélisation en l'appliquant à un autre type de système configurable, ainsi que de faire évoluer notre plate-forme d'acquisition de spécifications utilisateur en une interface interactive permettant de guider l'utilisateur dans la spécification de ses exigences en lui soumettant de manière interactive des retours précis sur les erreurs de spécifications identifiées et en simulant les règles de comportement validées. La réalisation de la spécification d'un environnement intelligent (*i.e.* géné-

ration et déploiement de la configuration sur les capteurs, actionneurs et processus de contrôle) représente, quant à elle, une deuxième perspective qui fera l'objet d'un travail futur.

Bibliographie

Al Balushi T. H., Sampaio P. R. F., Loucopoulos P. (2013). Eliciting and prioritizing quality requirements supported by ontologies: a case study using the elicito framework and tool. *Expert Systems*, vol. 30, n° 2, p. 129–151.

Albreshne A., Ait Lahcen A., Pasquier J. (2013). A framework and its associated process-oriented domain specific language for managing smart residential environments. *International Journal of Smart Home.*, vol. 7, n° 6, p. 377-392.

Avancha S., Patel C., Joshi A. (2004). Ontology-driven adaptive sensor networks. In *Proceedings of the first annual international conference on mobile and ubiquitous systems: Networking and services (mobiquitous'04)*, p. 194-202.

Bae I.-H. (2014). An ontology-based approach to {ADL} recognition in smart homes. *Future Generation Computer Systems*, vol. 33, n° 0, p. 32 - 41.

Bajwa I. S., Bordbar B., Lee M., Anastasakis K. (2012). Nl2alloy: A tool to generate alloy from nl constraints. *Journal of Digital Information Management*, vol. 10, n° 6.

Castañeda V., Ballejos L., Caliusco M. L., Galli M. R. (2010). The use of ontologies in requirements engineering. *Global Journal of Researches In Engineering*, vol. 10, n° 6.

Castañeda V., Ballejos L. C., Caliusco M. L. (2012). Improving the quality of software requirements specifications with semantic web technologies. In *Wer.*

Cherrier S., Ghamri-Doudane Y., Lohier S., Roussel G. (2011). D-lite: Distributed logic for internet of things services. In *Internet of things (ithings/cpscom), 4th international conference on cyber, physical and social computing*, p. 16-24.

Cherrier S., Ghamri-Doudane Y., Lohier S., Roussel G. *et al.* (2013). Salt: a simple application logic description using transducers for internet of things. In *Ieee international conference on communications icc.*

Chniti A., Albert P., Charlet J. (2012). Gestion de la cohérence des règles métier éditées à partir d'ontologies owl. In *Actes de ic2011*, p. 589-606.

Clavel M., Durán F., Eker S., Lincoln P., Marti-Oliet N., Meseguer J. *et al.* (2011). *Maude manual (version 2.6)* (vol. 1) n° 3. http://maude.cs.uiuc.edu/maude2-manual/maude-manual.pdf. (accédé : Décembre 2013)

Clavel M., Durán F., Hendrix J., Lucas S., Meseguer J., ölveczky P. (2007). *The maude formal tool environment* (vol. 4624).

Corno F., Sanaullah M. (2013). Modeling and formal verification of smart environments. *Security and Communication Networks*, vol. 7, n° 10, p. 1582–1598.

Du Bousquet L., Nakamura M., Yan B., Igaki H. (2009). Using formal methods to increase confidence in a home network system implementation: a case study. *Innovations in Systems and Software Engineering*, vol. 5, n° 3, p. 181-196.

Duràn F., Roldàn M. (2012). *Validating ocl constraints on maude prototypes of uml models.* http://maude.lcc.uma.es/mOdCL/docs/TR/DynamicValidation/TR-Prototyping.pdf. (accédé : Décembre 2013)

Eker S., Meseguer J., Sridharanarayanan A. (2004). The maude {LTL} model checker. *Electronic Notes in Theoretical Computer Science*, vol. 71, p. 162 - 187.

Fougères A.-J., Trigano P. (1997). Rédaction de spécifications formelles : Élaboration à partir des spécifications écrites en langage naturel. *Cognito - Cahiers Romans de Sciences Cognitives*, vol. 1, nº 8, p. 29-36.

Fraser M. D., Kumar K., Vaishnavi V. K. (1991). Informal and formal requirements specification languages: Bridging the gap. *IEEE Trans. Softw. Eng.*, vol. 17, nº 5, p. 454-466.

Gordon M., Harel D. (2009). Generating executable scenarios from natural language. In *Proceedings of the 10th international conference on computational linguistics and intelligent text processing (cicling '09)*, p. 456-467.

Guarino N. (1997). Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, vol. 46, nº 2-3, p. 293-310.

Guillet S., Bouchard B., Bouzouane A. (2013). Correct by construction security approach to design fault tolerant smart homes for disabled people. *Procedia Computer Science*, vol. 21, p. 257–264.

Guissé A. (2013). *Une plateforme daide à lacquisition et à la maintenance des règles métier à partir de textes règlementaires*. These, Université Paris 13 - Sorbonne Paris Cité.

Guissé A., Lévy F., Nazarenko A. (2012). From regulatory texts to brms: how to guide the acquisition of business rules? In *Proceedings of the 6th international conference on rules on the web: research and applications (ruleml'12)*, p. 77-91.

Horrocks I., Patel-Schneider P. F., Boley H., Tabet S., Grosof B., Dean M. (2004). *Swrl: A semantic web rule language combining owl and ruleml*. Rapport technique. World Wide Web Consortium.

Huang N., Wang X., Rocha C. (2009). Formal semantics of owl-s with rewrite logic. *JSEA*, vol. 2, nº 1, p. 25-33.

Karpovic J., Nemuraite L. (2011). Transforming sbvr business semantics into web ontology language owl2:main concepts. In *Proc. 17th international conference on information and software technologies.*

Khattak A. M., Truc P. T. H., Hung L. X., Vinh L. T., Dang V.-H., Guan D. *et al*. (2011). Towards smart homes using low level sensory data. *Sensors*, vol. 11, nº 12, p. 11581–11604.

Klein M., Schmidt A., Lauer R. (2007). Ontology-centred design of an ambient middleware for assisted living: The case of soprano. In *Towards ambient intelligence: Methods for cooperating ensembles in ubiquitous environments (aim-cu), 30th annual german conference on artificial intelligence.*

Kof L. (2009). Requirements analysis: concept extraction and translation of textual specifications to executable models. In *Proceedings of the 14th international conference on applications of natural language to information systems*, p. 79-90.

Körner S. J., Brumm T. (2009). Resi - a natural language specification improver. *IEEE Sixth International Conference on Semantic Computing*, vol. 0, p. 1-8.

Li P.-S., Liu A., Zhou P.-C. (2014, June). Context reasoning for smart homes using case-based reasoning. In *Consumer electronics (isce 2014), the 18th ieee international symposium*, p. 1-2.

Martí-Oliet N., Meseguer J. (2002). Rewriting logic as a logical and semantic framework. In *Handbook of philosophical logic*, vol. 9, p. 1-87.

Meziane F., Vadera S. (2010). Artificial intelligence in software engineering current developments and future prospects. In *Artificial intelligence applications for improved software engineering development: New prospects*, p. 2429.

Mich L., Franch M., Inverardi P. (2004). Market research for requirements analysis using linguistic tools. *Requirement Engineering*, vol. 9, n° 1, p. 40-56.

Njonko P., El Abed W. (2012). From natural language business requirements to executable models via sbvr. In *Systems and informatics (icsai), 2012 international conference on.*

O'Connor M. J., Das A. K. (2008). Sqwrl: A query language for owl. In *Owled*, vol. 529.

Omoronyia I., Sindre G., Stålhane T., Biffl S., Moser T., Sunindyo W. (2010). *A domain ontology building process for guiding requirements elicitation* (vol. 6182).

Petasis G., Karkaletsis V., Paliouras G., Krithara A., Zavitsanos E. (2011). Ontology population and enrichment: State of the art. In *Knowledge-driven multimedia information extraction and ontology evolution'11.*

Poli R. (1996). Ontology for knowledge organization. *Advances in Knowledge Organization*, vol. 5, p. 313-319.

Reijers N., Lin K.-J., Wang Y.-C., Shih C.-S., Hsu J. Y. (2013). Design of an intelligent middleware for flexible sensor configuration in m2m systems. In *Sensornets'13*, p. 41-46.

Reisig W. (1985). *Petri nets: An introduction*. Springer-Verlag New York.

Roche C. (2008). Le terme et le concept : fondements d'une ontoterminologie. *Terminologie & Ontologie : Theories et applications (TOTh).*

Roldan M., Duran F. (2010). Dynamic validation of ocl constraints with modcl. In *Workshop on ocl and textual modelling*, vol. 44.

Romero J. R., Rivera J. E., Durán F., Vallecillo A. (2007). Formal and tool support for model driven engineering with maude. *Journal of Object Technology*, vol. 6, n° 9, p. 187-207.

Ruiz-Martìnez J. M., Giménez J. A. Miñarro, Castellanos-Nieves D., Garcìa-Sánchez F., Valencia-Garcìa R. (2011). Ontology population: an application for the e-tourism domain. *International Journal of Innovative Computing; Information and Control (IJICIC)*, vol. 7, n° 11, p. 6115-6134.

Sadoun D. (2014). *Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot*. Theses, Université Paris Sud - Paris XI. Consulté sur https://tel.archives-ouvertes.fr/tel-01060540

Sadoun D., Dubois C., Ghamri-Doudane Y., Grau B. (2013a). From natural language requirements to formal specification using an ontology. In *25th international conference on tools with artificial intelligence (ictai).*

Sadoun D., Dubois C., Ghamri-Doudane Y., Grau B. (2013b). Peuplement d'une ontologie guidé par l'identification d'instances de propriété. In *Actes de la 10e conférence tia.*

Sadoun D., Dubois C., Ghamri-Doudane Y., Grau B. (2014). Formal rule representation and verification from natural language requirements using an ontology. In *Actes de la conférence ruleml'14, prague, république tchèque.*

Senanayake R., Denker G., Pearce J. (2005). Towards integrated specification and analysis of machine-readable policies using maude 1. In *Workshop on semantic web and policies at international semantic web conference.*

Sirin E., Parsia B., Grau B., Kalyanpur A., Katz Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics Science Services and Agents on the World Wide Web*, vol. 5, n° 2, p. 51-53.

Song Y., Chen R., Liu Y. (2012). A non-standard approach for the owl ontologies checking and reasoning. *Journal of Computers (JCP)*, vol. 7, n° 10, p. 2454-2461.

Sukys A., Nemuraite L., Paradauskas B., Sinkevicius E. (2012). Transformation framework for sbvr based semantic queries in business information systems. In *The second international conference on business intelligence and technology*, p. 19-24.

Szulman S. (2011). Une nouvelle version de l'outil terminae de construction de ressources termino-ontologiques. In *22èmes journées francophones d'Ingénierie des Connaissances (poster)*, p. 3 pages.

Thongkrau T., Lalitrojwong P. (2012). Ontopop: An ontology population system for the semantic web. *IEICE Transactions*, vol. 95, n° 4, p. 921-931.

Vadera S., Meziane F. (1994). From english to formal specifications. *The Computer Journal*, vol. 37, n° 9, p. 753-763.

Verdejo A., Martí-Oliet N., Robles T., Salvachúa J., Llana L., Bradley M. (2005). Transforming information in rdf to rewriting logic. In *Formal methods for open object-based distributed systems*, vol. 3535, p. 227-242.

Wang F., Turner K. J. (2009). An ontology-based actuator discovery and invocation framework in home care systems. In *Proceedings of the 7th international conference on smart homes and health telematics: Ambient assistive health and wellness management in the heart of the city*, p. 66-73.

Wei M., Xu J., Yun H., Xu L. (2012). Ontology-based home service model. *Computer Science and Information Systems*, vol. 9, n° 2, p. 813–838.

Xu J., Lee Y.-H., Tsai W.-T., Li W., Son Y.-S., Park J.-H. *et al.* (2009). Ontology-based smart home solution and service composition. In *Embedded software and systems, 2009. icess '09. international conference on*, p. 297-304.

Yan B., Nakamura M., Bousquet L. du, Matsumoto K. ichi. (2008). Validating safety for the integrated services of the home network system using jml. *JIP*, p. 38-49.