Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot

Driss Sadoun

École Doctorale : ED 427

École doctorale d'Informatique de Paris-Sud (EDIPS)

Laboratoire : LIMSI-CNRS

Laboratoire d'Informatique Mécanique et Sciences de l'Ingénieur (UPR 3251)

THÈSE DE DOCTORAT EN INFORMATIQUE

Driss Sadoun

Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot

---

Soutenue le 17 Juin 2014

Composition du jury

Co-directeurs :	M^me^	Brigitte GRAU	**Professeur** (LIMSI-CNRS, ENSIIE)
	M^me^	Catherine DUBOIS	**Professeur** (CEDRIC-CNAM, ENSIIE)
	M.	Yacine GHAMRI-DOUDANE	**Professeur** (L3i, Université de La Rochelle)
Rapporteurs **:**	M.	François LÉVY	**Professeur** (LIPN, Université Paris Nord)
	M.	Yamine AIT-AMEUR	**Professeur** (IRIT, ENSEEIHT)
Examinateurs **:**	M^me^	Chantal REYNAUD	**Professeur** (LRI, Université Paris-Sud)
	M.	Yannick TOUSSAINT	**CR HDR** (LORIA-INRIA)

RÉSUMÉ

Le développement d'un système a pour objectif de répondre à des exigences. Aussi, le succès de sa réalisation repose en grande partie sur la phase de spécification des exigences qui a pour vocation de décrire de manière précise et non ambiguë toutes les caractéristiques du système à développer. Les spécifications d'exigences sont le résultat d'une analyse des besoins faisant intervenir différentes parties. Elles sont généralement rédigées en langage naturel (LN) pour une plus large compréhension, ce qui peut mener à diverses interprétations, car les textes en LN peuvent contenir des ambiguïtés sémantiques ou des informations implicites. Il n'est donc pas aisé de spécifier un ensemble complet et cohérent d'exigences. D'où la nécessité d'une vérification formelle des spécifications résultats. Les spécifications LN ne sont pas considérées comme formelles et ne permettent pas l'application directe de méthodes vérification formelles. Ce constat mène à la nécessité de transformer les spécifications LN en spécifications formelles. C'est dans ce contexte que s'inscrit cette thèse. La difficulté principale d'une telle transformation réside dans l'ampleur du fossé entre spécifications LN et spécifications formelles. L'objectif de mon travail de thèse est de proposer une approche permettant de vérifier automatiquement des spécifications d'exigences utilisateur, écrites en langage naturel et décrivant le comportement d'un système. Pour cela, nous avons exploré les possibilités offertes par un modèle de représentation fondé sur un formalisme logique. Nos contributions portent essentiellement sur trois propositions : 1) une ontologie en OWL-DL fondée sur les logiques de description, comme modèle de représentation pivot permettant de faire le lien entre spécifications en langage naturel et spécifications formelles ; 2) une approche d'instanciation du modèle de représentation pivot, fondée sur une analyse dirigée par la sémantique de l'ontologie, permettant de passer automatiquement des spécifications en langage naturel à leur représentation conceptuelle ; et 3) une approche exploitant le formalisme logique de l'ontologie, pour permettre un passage automatique du modèle de représentation pivot vers un langage de spécifications formelles nommé Maude.

ABSTRACT

The main objective of system development is to address requirements. As such, success in its realisation is highly dependent on a requirement specification phase which aims to describe precisely and unambiguously all the characteristics of the system that should be developed. In order to arrive at a set of requirements, a user needs analysis is carried out which involves different parties (stakeholders). The system requirements are generally written in natural language to garantuee a wider understanding. However, since NL texts can contain semantic ambiguities, implicit information, or other inconsistenties, this can lead to diverse interpretations. Hence, it is not easy to specify a set of complete and consistent requirements, and therefore, the specified requirements must be formally checked. Specifications written in NL are not considered to be formal and do not allow for a direct application of formal methods. We must therefore transform NL requirements into formal specifications. The work presented in this thesis was carried out in this framework. The main difficulty of such transformation is the gap between NL requirements and formal specifications.

The objective of this work is to propose an approach for an automatic verification of user requirements which are written in natural language and describe a system's expected behaviour. Our approach uses the potential offered by a representation model based on a logical formalism.

Our contribution has three main aspects : 1) an OWL-DL ontology based on description logic, used as a pivot representation model that serves as a link between NL requirements to formal specifications ; 2) an approach for the instantiation of the pivot ontology, which allows an automatic transformation of NL requirements to their conceptual representations ; and 3) an approach exploiting the logical formalism of the ontology in order to automatically translate the ontology into a formal specification language called Maude.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

ACRONYMES

AA : apprentissage automatique.
AIA : Accord inter-annotateurs.
ACE : Attempto Controlled English.
EI : Extraction d'Information.
IA : Intelligence Artificielle.
IC : Ingénierie des Connaissances.
LF : Langage Formel.
LN : Langage Naturel.
LNC : Langage Naturel Contrôlé.
LSC : Live Sequence Chart.
MOF : Meta-Object Facility.
MSC : Message Sequence Chart.
OCL : Object Constraint Language.
OMG : Object Management Group.
OWL : Web Ontology Language.
RCEN : Reconnaissance et la Classification d'Entités Nommées.
RDF : Resource Description Framework.
RDFS : Resource Description Framework Schema.
SBVR : Semantics of Business Vocabulary and Business Rules.
SQWRL : Semantic Query-Enhanced Web Rule Language.
SWRL : A Semantic Web Rule Language.
SysML : Systems Modeling Language.
TAL : Traitement Automatique des Langues.
UML : Unified Modeling Language.

INTRODUCTION

> « When Thomas Edison worked late into the night on the electric light, he had to do it by gas lamp or candle. I'm sure it made the work seem that much more urgent. »
>
> **George Carlin**

CONTEXTE

De nos jours, un intérêt croissant est observé à l'égard des nouvelles technologies, notamment informatiques, pour résoudre des problèmes de la vie courante. Bien que courte, l'histoire de l'informatique compte de très nombreuses révolutions. Elle est d'abord marquée par l'apparition du *calculateur*, machine qui permet d'effectuer des calculs numériques complexes. Le *calculateur* est très vite remplacé par l'*ordinateur* qui étend ses capacités au traitement de données universelles. Une des premières formes de l'ordinateur est celui dit central, ou *mainframe*, offrant une grande puissance de traitement destinée aux entreprises et aux centres de recherche. Le mini-ordinateur succède au *mainframe*, moins puissant mais meilleur marché, il est aujourd'hui plus communément appelé *serveur*. Comme une suite logique le micro-ordinateur, ou ordinateur personnel, apparaît. Moins coûteux, il est à la portée de chacun. L'apparition du web offre encore un nouveau tournant à l'informatique ; l'ordinateur n'est plus seulement outil de calcul mais de communication et d'échange, entre individus, entre machines, voire entre individus et machines. Enfin, l'informatique contemporaine est ubiquitaire. L'ordinateur d'aujourd'hui s'efface et devient logiciel, on l'embarque sans le voir, on l'utilise sans y penser, on le configure sans le comprendre. Cette omniprésence laisse imaginer des applications qui ne s'adaptent plus seulement à la société mais à la personne. Les systèmes informatiques d'aujourd'hui doivent alors s'adapter aux besoins spécifiques de chacun.

Le développement d'un système a pour objectif de répondre à des exigences. Aussi, le succès de sa réalisation repose en grande partie sur la phase de spécification. Cette phase a pour objectif d'identifier les exigences que le système réalisé devra satisfaire. La spécification des exigences a pour vocation de décrire de manière précise

et non ambiguë toutes les caractéristiques du système à développer (son comportement, ses propriétés et contraintes), afin de guider et de faciliter sa réalisation.

Le document appelé classiquement « spécifications des besoins » doit documenter l'ensemble des exigences de façon claire, compréhensible et souvent contractuelle. Un défaut de spécification peut donc mener au rejet du système par le client. De plus, une erreur non identifiée dans les spécifications engendre fatalement un coût supplémentaire dû à une correction tardive et peut, dans le cas de systèmes critiques, mener à des catastrophes humaines, économiques ou sociales.

Les spécifications d'exigences sont le résultat d'une analyse des besoins faisant intervenir différentes parties. Cette analyse se base sur l'étude de l'existant éventuel, l'énoncé des besoins exprimés par l'utilisateur et les connaissances liées au domaine d'application, souvent obtenues auprès d'experts. Il n'est pas toujours garanti que toutes les parties aient une compréhension commune et complète du problème. En effet, l'utilisateur n'a pas toujours une idée précise et complète du problème et les experts, selon leur domaine de compétence, peuvent parler des langages différents. Il n'est donc pas aisé de spécifier un ensemble complet et cohérent d'exigences. Les exigences ont alors tendance à évoluer en même temps que le développement du système et la compréhension de ses objectifs. L'amélioration des spécifications d'exigences a motivé de nombreux travaux scientifiques. Cette amélioration passe entre autres par la vérification des spécifications. Elles doivent alors être représentées de telle sorte à pouvoir être traitées par des méthodes de vérification formelles. C'est dans ce contexte que s'inscrit cette thèse.

Les méthodes formelles sont définies par une syntaxe et une sémantique précise, ainsi qu'un mécanisme de raisonnement formel. Elles permettent de raisonner rigoureusement sur des spécifications formelles et de démontrer leur validité. Ces méthodes permettent d'obtenir une très forte assurance quant à la cohérence des spécifications décrivant le système à réaliser.

PROBLÉMATIQUE

L'objectif de ce travail est de proposer une approche réaliste et pragmatique permettant de vérifier automatiquement des spécifications d'exigences utilisateur, écrites en langage naturel (LN) et décrivant le comportement d'un système. Si l'utilisation des langages naturels permet une compréhension plus large des différentes parties, elle peut cependant mener à diverses interprétations, car les textes en langage naturel peuvent contenir des ambiguïtés sémantiques ou des informations implicites. Les spécifications en langage naturel ne sont donc

pas considérées comme formelles et ne permettent pas l'application directe de méthodes formelles.

Ce constat mène à la nécessité de transformer les spécifications en langage naturel en spécifications formelles. Cette transformation est généralement coûteuse en ressources humaines et matérielles, ce qui cantonne actuellement l'application de méthodes formelles aux systèmes critiques. La difficulté réside principalement dans l'ampleur du fossé entre spécifications en langage naturel et spécifications formelles.

Les questions qui se posent alors sont :

- Comment identifier à partir des spécifications en langage naturel, les règles de comportement spécifiées ?
- Comment produire une représentation formelle du comportement d'un système ?
- Comment s'assurer de la cohérence et de la conformité du comportement décrit par les spécifications ?

Les réponses à ces questions doivent permettre de traiter l'ambiguïté et l'implicite inhérents aux spécifications en langage naturel avec l'impératif de préserver leur sémantique. Les solutions proposées devront dans l'idéal s'appliquer à différents domaines et s'adapter aisément à différentes langues.

CONTRIBUTIONS

Au travers de cette thèse, nous tentons de répondre à une question importante, à laquelle les chercheurs en intelligence artificielle et particulièrement en traitement automatique des langues et représentation des connaissances se sont intéressés de manière récurrente : comment les modèles de représentation et de formalisation peuvent-ils aider à écrire de meilleures exigences ? Et de façon plus générale, comment ces modèles peuvent-ils permettre de combler le fossé entre langage naturel et langage formel ? Pour formaliser un énoncé en langage naturel, il est préférable de définir son sens avec rigueur et de le représenter de façon suffisamment formelle pour y appliquer un raisonnement. La discipline qui traditionnellement s'intéresse à cette problématique est la logique.

Nous proposons donc d'explorer les possibilités offertes par un modèle de représentation fondé sur un formalisme logique. Nos contributions portent essentiellement sur trois propositions :

- une ontologie fondée sur les logiques de description, comme modèle de représentation pivot permettant de faire le lien entre spécifications en langage naturel et spécifications formelles ;
- une approche d'instanciation du modèle de représentation pivot, fondée sur une analyse dirigée par la sémantique de l'ontologie,

permettant de passer automatiquement des spécifications en langage naturel à leur représentation conceptuelle ;
– une approche exploitant le formalisme logique de l'ontologie, pour permettre un passage automatique du modèle de représentation pivot vers un langage de spécifications formelles.

Notre première contribution porte sur la proposition d'une ontologie comme modèle de représentation pivot, ayant pour but de faciliter le passage de spécifications LN vers des spécifications formelles. Nous avons choisi de définir ce modèle à l'aide d'une ontologie, car elle permet de représenter de manière formelle les connaissances d'un domaine. En outre, elle fournit un cadre de représentation et une sémantique formels, qui permettent de guider l'identification d'exigences à partir des spécifications et d'effectuer la vérification de celles-ci. Une ontologie a une syntaxe et une sémantique bien définies ainsi que des mécanismes de raisonnement et de vérification. L'ontologie proposée est décrite en OWL-DL une version décidable du langage d'ontologie web OWL. OWL-DL est fondé sur les logiques de description. Ces logiques divisent les connaissances en deux parties, une première partie terminologique qui définit des classes d'individus (des concepts), leurs propriétés et leurs contraintes. Ces connaissances sont génériques et s'appliquent à tous les individus dans tous les modèles. La seconde partie est assertionnelle, elle définit des connaissances spécifiques à certains individus. Nous proposons de mettre à profit ces deux niveaux de connaissances, pour décrire à partir des connaissances du domaine, un modèle générique du comportement d'un système, qui sera spécialisé à partir de l'analyse des spécifications des besoins utilisateur. L'usage d'une ontologie formelle permet de définir un ensemble de notations et de contraintes, suffisamment précises et formelles pour supposer qu'elle puisse se substituer aux experts du domaine [1].

Notre seconde contribution correspond à la définition d'une approche permettant le passage des spécifications en langage naturel vers l'ontologie. Pour cela, nous partons d'une ontologie de haut niveau, modélisant le comportement générique d'un système. Ce modèle est ensuite spécialisé grâce au peuplement de l'ontologie, qui s'effectue de manière automatique à partir des spécifications utilisateur. Peupler une ontologie consiste à y ajouter un ensemble d'instances de concepts (individus) et de propriétés. Dans notre contexte, les individus correspondent à des objets du monde. En général, les objets peuvent avoir plusieurs dénominations, ils se distinguent alors par leurs caractéristiques. Nous proposons une approche de peuplement d'ontologie guidée par l'identification d'instances de propriétés.

1. A good notation has a subtlety and suggestiveness which at times make it seem almost like a live teacher. *Bertrand Russell. Tractatus Logico-Philosophicus* (1956)

Les instances de propriétés sont extraites à l'aide de règles d'extraction acquises automatiquement à partir de leurs différents contextes syntaxico-sémantiques de formulation dans les textes. L'originalité de cette approche est qu'elle fonde l'identification des individus non pas sur leurs mentions dans les spécifications qui peuvent être ambiguës, ou même implicites, mais sur leurs valeurs de propriétés identifiées dans les textes. En effet, un individu est caractérisé par les valeurs des propriétés qui lui sont associées. La reconnaissance de celles-ci peut mener à une identification précise et fiable d'un individu. Par exemple, selon les choix de modélisation [2] 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.

L'ontologie, de par son formalisme logique, permet d'effectuer un certain nombre de vérifications. Cependant l'ensemble des vérifications que nous souhaitons effectuer ne sont pas toutes possibles sous OWL. En effet, le raisonnement OWL est monotone [3]. Il 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. Néanmoins, le formalisme logique de l'ontologie permet un passage automatique de notre modèle de représentation pivot vers un langage de spécifications formelles, offrant des vérifications complémentaires de celles offertes sous OWL. Nous montrons qu'à partir du modèle ontologique conçu, il est aisé de passer vers un tel langage. Le choix de ce langage doit être motivé en premier lieu par la complémentarité des vérifications qu'il offre par rapport à celles possibles sous OWL. De plus, le langage de spécification formelle doit permettre de représenter et de vérifier l'évolution de l'état d'un système en fonction de ses règles de comportement. Notre choix s'est porté sur *Maude*, un langage de spécifications formelles, fondé sur la logique de réécriture, car il est bien adapté à la spécification et la vérification de systèmes à base de règles concurrentes. Notre dernière contribution consiste alors en une méthode permettant

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

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

la transformation de notre modèle ontologique vers une spécification formelle *Maude*.

Enfin, nous montrons que tout au long de ce processus de formalisation des spécifications d'exigences, il est possible de vérifier et d'assurer la complétude, la cohérence et la conformité des exigences spécifiées.

Notre contribution globale est la proposition d'une approche, permettant de faciliter les phases de spécification formelle et de vérification d'exigences, ce qui présente un intérêt tant scientifique qu'industriel.

Plan du document

Ce manuscrit présente au chapitre 1 un état de l'art, portant sur la formalisation de spécifications d'exigences écrites en langage naturel. Nous y décrivons les enjeux, les problématiques, les propositions des différentes communautés scientifiques ainsi que notre positionnement par rapport à ces propositions. Nous concluons ce chapitre par la description de notre approche. Le chapitre 2 fournit une description usuelle puis formelle d'une ontologie, puis fait un état de l'art des méthodes et approches de conception d'ontologie et plus particulièrement de peuplement d'ontologie. Dans le chapitre 3, nous présentons et motivons notre modélisation d'une ontologie générique du comportement d'un système et détaillons notre approche de peuplement d'ontologie fondée sur l'exploitation des caractéristiques de notre modèle. En outre, nous présentons les vérifications permises par le modèle ontologique obtenu. Notre proposition de passage du modèle ontologique instancié vers le langage de spécification formelle *Maude* est détaillée dans le chapitre 4. Nous y présentons aussi les vérifications supplémentaires qui complètent celles autorisées par OWL. Dans le chapitre 5, nous positionnons le cadre dans lequel s'inscrit cette thèse, à savoir, le projet ENVIE VERTE[4] dont l'objectif est de permettre à un utilisateur de configurer son environnement intelligent, simplement en décrivant textuellement ses besoins en langage naturel. Suite à la description de notre cas d'application, qui porte sur le domaine des environnements intelligents, des ressources et outils mis en place pour nos expérimentations, nous présentons et discutons les résultats obtenus. Enfin, le document se termine par une conclusion globale et la présentation des perspectives.

4. financé par DIGITEO, projet DIM LSC 2010.

1

FORMALISATION DE SPÉCIFICATIONS D'EXIGENCES

Sommaire

> « When I use a word, it means just what I choose it to mean - neither more nor less. »
>
> **Humpty Dumpty in Lewis Carroll's Through the Looking-Glass**

Dans ce chapitre, nous présentons la problématique du passage automatique de spécifications en langage naturel vers des spécifications formelles. Nous commençons par présenter les enjeux ainsi que les difficultés, puis donnons un aperçu des efforts scientifiques menés dans cette voie et pour finir, nous décrivons notre approche avant de conclure.

1.1 INTRODUCTION

De nos jours, le système est omniprésent, en constante évolution et de plus en plus sophistiqué. Le nombre de fonctionnalités et d'options qui l'accompagnent n'a de cesse d'augmenter. Face à ce constat, les contraintes du marché imposent de produire toujours plus vite et moins cher. L'ensemble peut mener à une baisse de qualité des technologies et de leur maintien. Le nombre d'exigences relatives à un nouveau système augmente lui aussi avec la complexité des systèmes modernes. Dans le cas d'un système critique, ses exigences doivent être formalisées de sorte à pouvoir être ensuite vérifiées. La formalisation est une tâche difficile, principalement lorsque les spécifications sont complexes [Ilic, 2007].

Lors de la conception d'un système, il est recommandé d'identifier de façon précise les objectifs, fonctionnalités et contraintes [Peres *et al.*, 2012]. Au travers de divers échanges et discussions, les différentes parties expriment leurs besoins et contraintes. La documentation relative à ces besoins et contraintes décrit les attentes liées au système. Les documents produits lors de cette phase correspondent aux spécifications d'exigences. Sommerville et Sawyer [1997] définissent les exigences comme un cahier des charges de ce qui devrait être mis en œuvre. Elles décrivent la façon dont le système doit se comporter, les propriétés et attributs du système, ainsi que les contraintes de développement. Pour *Gordon* et *Harel* [Gordon et Harel, 2009], les exigences sont une façon de décrire les scénarios qui doivent se produire, ceux qui peuvent se produire, et ceux qui ne sont pas autorisés à se produire.

La spécification d'exigences est un pré-requis pour les étapes de modélisation et de développement d'un système. Les spécifications d'exigences produites ont comme objectif de décrire le système de manière consensuelle entre les "parties prenantes", et d'orienter la construc-

tion du système. Ces spécifications doivent être non-ambiguës, maintenues à jour et vérifiées tout au long du projet.

La modélisation du système et de ses différentes fonctions s'effectue donc à partir des spécifications d'exigences. Afin de produire un modèle cohérent, il est nécessaire de vérifier la cohérence et la complétude des spécifications d'exigences qui en sont le point de départ. Les étapes de spécification, de modélisation et de vérification des exigences sont étroitement liées et doivent se combiner de façon itérative. Le succès du développement d'un système dépend de la qualité des spécifications d'exigences. De mauvaises exigences mènent à de mauvais systèmes. Aussi, l'amélioration de la qualité des systèmes à concevoir passe entre autres par l'amélioration des spécifications qui le décrivent.

Selon Wiegers [Wiegers, 2000] un projet doit comprendre trois niveaux d'exigences :

- Les exigences non fonctionnelles : elles décrivent pourquoi le produit est en cours de construction et identifient les bénéfices que les clients et l'entreprise vont récolter ;
- Les exigences clients, capturées sous la forme de cas d'utilisation, décrivent les tâches et le domaine ;
- Les exigences fonctionnelles : elles décrivent les comportements spécifiques du système qui doit être mis en œuvre. Ces exigences peuvent dépendre des connaissances du domaine ou être plus spécifiques aux besoins des utilisateurs.

Dans le cadre de cette thèse, nous nous intéressons aux exigences fonctionnelles.

L'ingénierie des exigences consiste à comprendre ce que l'on a l'intention de construire avant d'avoir fini de le construire [Wiegers, 2000]. L'ingénierie des exigences repose essentiellement sur la communication entre les différents participants. L'objectif principal est de permettre une compréhension commune, par les différents intervenants du projet, du système à concevoir. En général, cette compréhension est représentée sous la forme de documents écrits et de modèles graphiques, ce qui correspond à la spécification des exigences. Durant une phase de conception, des langages de différents niveaux peuvent être utilisés ; leur utilisation dépend alors de la phase du projet et de la tâche (compréhension, implantation ou analyse)

1.2 LANGAGES DE SPÉCIFICATION DES EXIGENCES

Les langages de spécifications d'exigences peuvent être classés en deux catégories : les langages non formels, et les langages formels [Fraser *et al.*, 1991]. Les langages de spécification non formels renvoient aux spécifications décrites en langage naturel. Les langages

naturels contrôlés peuvent être considérés comme une troisième catégorie, se positionnant entre langages non formels et formels.

1.2.1 *Langage formel*

Les langages formels (LF) obéissent à une syntaxe et une sémantique toutes deux bien définies, dans un style souvent mathématique. Ils sont conçus pour être les moins ambigus possibles. Ils tendent à être concis et peu redondants. Chaque terme ou formulation ne peut être interprété que d'une seule manière, ce qui permet d'expliciter des informations de manière précise et sans ambiguïté.

La complexité de leur syntaxe et, plus encore, leurs fondements mathématiques, les rendent difficilement compréhensibles par une personne non experte. Aussi, les deux activités d'écriture et de lecture ne sont pas simples et requièrent un certain degré d'expertise. La plupart des langages formels sont construits de manière à répondre à une problématique spécifique, et ne sont donc pas aptes à modéliser de façon naturelle (sans de lourdes modifications) l'ensemble des spécifications d'un système.

Leur principal avantage est que les machines peuvent les comprendre et les interpréter. Cela autorise l'application de processus automatique d'analyse, de vérification, de transformation, etc.

1.2.2 *Langage naturel*

Un langage naturel (LN) n'est pas non formel en lui-même. Il est défini par un ensemble de règles (de grammaire) et sur un vocabulaire fini bien que très large. Néanmoins, ce vocabulaire est évolutif et les pratiquants d'une langue ne s'accordent pas toujours sur les règles grammaticales à respecter (ou à appliquer) ni sur la sémantique qui leur est associée.

Le langage naturel est le langage qui est parlé ou écrit (français, anglais ...) pour communiquer. Tout langage naturel est facilement utilisable et compréhensible par un humain parlant ce langage, ce qui en fait le moyen le plus simple et direct pour communiquer et exprimer ses besoins ou la perception d'un problème.

Le langage naturel est sans doute le langage le plus expressif. Son écriture donne la possibilité d'exprimer une même idée en la formulant de différentes manières et d'avoir pour une même formulation plusieurs sens. Cela peut donner lieu à des problèmes de communication dus à des interprétations multiples. La compréhension du sens d'un énoncé demande parfois au lecteur, un effort d'interprétation ou d'inférence en fonction d'un contexte particulier. La compré-

hension du sens fait souvent appel à des connaissances implicites et extérieures à l'énoncé. En outre, dans le cas d'une collaboration en domaine pluridisciplinaire, la terminologie utilisée peut différer et donc poser des problèmes de communication et de compréhension.

La richesse des langages naturels, ainsi que l'interprétation variable du sens terminologique et grammatical des formulations possibles font que les spécifications écrites en langage naturel sont connues pour être ambiguës. Les spécifications LN sont aussi connues pour pouvoir véhiculer des informations pertinentes de façon implicite. Dans ce cas, l'appel à des connaissances extérieures aux textes ainsi qu'un effort de déduction sont nécessaires pour la compréhension de la spécification et peuvent malgré tout conduire à une mauvaise interprétation.

Du fait de leurs ambiguïtés et de leur contenu implicite, les spécifications en LN posent le problème d'être difficilement interprétables par une machine. Il n'est donc pas possible d'effectuer de manière directe une vérification automatique prenant ce genre de spécification en entrée.

1.2.3 *Langage naturel contrôlé*

Un langage naturel contrôlé (LNC) est un langage construit à partir d'un langage naturel, correspondant à un sous-ensemble d'un LN selon Schwitter [2010]. Kuhn [2014] juge l'utilisation du terme sous-ensemble trompeuse, car de nombreux LNC ne sont pas des sous-ensembles stricts d'un langage naturel : beaucoup font de petits écarts par rapport à la grammaire ou la sémantique initiale, tandis que d'autres font appel à des éléments non présents dans le langage naturel tels que la coloration ou des caractères spéciaux pour augmenter la lisibilité et la précision. *Kuhn* donne la définition suivante : *Un langage naturel contrôlé est un langage construit qui est fondé sur un langage naturel, mais est plus restrictif concernant le lexique, la syntaxe et / ou la sémantique, tout en conservant l'essentiel de ses propriétés naturelles.* « *A controlled natural language is a constructed language that is based on a certain natural language, being more restrictive concerning lexicon, syntax and/or semantics while preserving most of its natural properties.* » Le niveau de formalisation varie d'un langage à l'autre en fonction de l'utilisation qui en est faite. Dans son étude sur la classification des langages naturels contrôlés [Kuhn, 2014], *Kuhn* a listé entre 1930 et aujourd'hui, pas moins de 99 langages contrôlés, fondés sur l'anglais tels que Formalized-English, ACE ou SBVR.

Ces langages contraignent la syntaxe et restreignent le lexique afin de réduire la difficulté d'interprétation des langages naturels. La syntaxe autorisée est simplifiée et le lexique est réduit à un sous ensemble.

Leur utilisation a pour but, d'une part d'améliorer la communication entre les êtres humains, en fournissant des représentations naturelles et intuitives et d'autre part, de simplifier la représentation, l'acquisition et le traitement de connaissances afin de faciliter une transition automatique vers une représentation formelle. Néanmoins, l'écriture demande un apprentissage des règles de restriction de la grammaire et du lexique autorisé ou l'utilisation d'outils d'aide à l'écriture de LNC.

1.3 INTERACTION ENTRE LANGAGES

La spécification d'exigences est un travail collaboratif entre le client, qui seul connaît réellement le problème et les concepteurs qui doivent l'aider à l'exprimer correctement [Ambriola et Gervasi, 1997a]. Ces exigences doivent être simples et spécifiées dans un langage commun, que les différentes parties du projet, experts et non experts seront aptes à comprendre. Cela écarte l'utilisation des langages formels, qui sont moins largement connus et requièrent un certain niveau d'expertise.

Les spécifications des exigences sont donc généralement écrites en langage naturel, car cela permet une plus large compréhension entre les différentes parties prenantes d'un même projet [Peres *et al.*, 2012]. Effectivement, comme l'a montré l'enquête menée dans [Mich *et al.*, 2004], la majorité des documents disponibles pour la spécification des besoins sont fournis par le client en langage naturel.

En revanche, le manque de rigueur de ces documents et l'ambiguïté inhérente au langage naturel limitent la portée de leur utilisation. En effet, l'indéterminisme de leur interprétation peut mener à des différences de compréhension entre les différentes parties d'un même projet et donc mener à des problèmes de modélisation.

L'utilisation de langages formels qui sont plus précis et concis, permet de remédier à ces inconvénients. Les spécifications formelles constituent donc un meilleur point de départ pour les étapes ultérieures du cycle de vie (analyse, vérification, implémentation). Il est plus aisé d'éviter les erreurs de conception en passant de spécifications formelles à une implémentation [Andrews et Gibbins, 1988].

Quel que soit le langage utilisé, seul l'utilisateur sera habilité à valider les exigences dans leur forme finale. Il doit donc lui être possible de les comprendre, de les évaluer et de les valider [Ambriola et Gervasi, 1997b]. La forme finale des spécifications que l'utilisateur aura à valider est en général en langage naturel, car il sera rarement à même d'évaluer des spécifications en langage formel. La validation des spécifications formelles et de la correspondance entre non formel et for-

mel devra être prise en charge par les concepteurs (avec de préférence la participation de l'utilisateur ou du client).

En résumé, lors de la conception d'un système, les différents niveaux de formalisation de spécifications sont complémentaires. Mener à terme un projet impose l'emploi itératif de ces deux formalismes dans un processus évolutif qui adopte représentations non formelles, semi-formelles et formelles des connaissances. Selon Kitapci et Boehm [2006], gérer les transitions entre ces représentations est la clé du succès de toute tâche d'ingénierie.

1.4 FORMALISATION DE SPÉCIFICATIONS EN LANGAGE NATUREL

La formalisation de spécifications en langage naturel permet d'avoir à disposition une représentation interprétable et vérifiable automatiquement par une machine, de sorte à faire apparaître les ambiguïtés, omissions et contradictions liées à la prise en compte de l'ensemble des exigences spécifiées d'un même système.

Le passage de spécifications non formelles à des spécifications formelles se fait en général grâce à une intervention humaine. Dans ce cas, seules l'expertise et l'expérience peuvent aider à produire une formalisation correcte [Peres *et al.*, 2012]. Néanmoins, le nombre d'experts est restreint, comparé à la quantité grandissante de spécifications d'exigences produites qui accompagne le nombre croissant de systèmes à réaliser. De plus, les exigences en relation ou en concurrence ne se trouvent pas toujours proches dans un texte, ce qui complique d'autant plus la tâche pour une lecture humaine qui a tendance à traiter et analyser les textes de façon séquentielle. Dans ce contexte, il s'impose de faire appel à des approches limitant l'implication humaine de sorte à accélérer et améliorer le processus de formalisation. L'automatisation de ce processus a pour objectif de traiter un plus grand nombre de spécifications en un temps réduit et de manière moins subjective qu'un expert qui se fonde uniquement sur ses connaissances et compétences.

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 [Abbott, 1983; Saeki *et al.*, 1989; Fraser *et al.*, 1991; Ambriola et Gervasi, 1997a; Fougères et Trigano, 1997]. Elle a suscité et suscite encore un grand intérêt car elle donne lieu à de nombreux challenges dans différents domaines scientifiques tels que l'ingénierie des exigences, la représentation des connaissances, le traitement automatique de la langue ou la vérification formelle. Les avancées dans différents sous-domaines de l'intelligence artificielle (IA) tels que le traitement automatique des

langues (TAL), l'extraction d'information (EI), l'apprentissage automatique (AA) et l'ingénierie des connaissances (IC) permettent déjà d'automatiser des parties substantielles de cette tâche. Les recherches portant sur l'application de techniques de l'IA à l'ingénierie des exigences ont connu une croissance très importante au cours des deux dernières décennies [Meziane et Vadera, 2010]. L'utilisation des outils d'IA laisse ainsi espérer des logiciels développés plus rapidement et de meilleure qualité [Ammar *et al.*, 2012].

1.5 INTERPRÉTATION DE TEXTES EN LANGAGE NATUREL ET REPRÉSENTATION DES CONNAISSANCES

Au fil du temps, les langages naturels ont évolué plus pour répondre au besoin de communication qu'au besoin de représentation [Russell *et al.*, 1996]. Cette évolution nous amène dans une situation où la connaissance nécessaire à l'interprétation d'un texte n'est pas systématiquement énoncée de manière explicite. Le langage naturel permet de laisser implicite ce qui est supposé connu. Afin de faciliter la communication, des informations ne sont généralement pas données lorsque l'inférence de celles-ci est supposée possible. Cela permet de limiter la quantité des connaissances à expliciter dans les textes. Ces connaissances implicites doivent être inférées à partir de connaissances propres au domaine de référence.

L'interprétation d'un énoncé en langage naturel consiste à passer de sa forme littérale à son sens. À l'inverse des langages formels, la transition entre forme littérale et sens n'est pas mécanique. Un énoncé en langage naturel peut avoir plusieurs significations (ambiguïté) et, inversement, plusieurs énoncés sont possibles pour exprimer une même signification (paraphrase). Certaines connaissances peuvent ne pas être explicitées (implicite) et le sens de l'énoncé peut ne pas se suffire à lui-même (incomplet). L'ensemble de ces problèmes rendent l'interprétation d'un énoncé en LN difficile. C'est pourquoi une interprétation correcte ne peut toujours dépendre uniquement des connaissances propres aux textes.

Le sens d'un texte se construit à partir de la signification de ses unités linguistiques et de leurs combinaisons, ce qui relève du domaine de la sémantique. Un texte en langage naturel est une suite de mots qui s'organisent en phrases. Un mot (ou une combinaison de mots) forme une unité linguistique qui, lorsqu'elle appartient à un vocabulaire spécialisé est appelée *terme*. Néanmoins, les termes sont composés de mots qui peuvent posséder plusieurs significations. Il en résulte que les termes peuvent dénoter plusieurs significations ce qui les rend ambigus. Par exemple, le terme *light* peut désigner en anglais une lampe, un rayonnement ou l'action d'allumer. De fait, l'in-

terprétation d'un terme ne peut se limiter à la reconnaissance de sa forme littérale. Dans les textes, les termes entretiennent des relations sémantiques. La prise en compte de ces relations peut permettre de restreindre leur domaine d'interprétation et ainsi d'améliorer l'interprétation globale du texte. Par exemple, dans la phrase *Kitchen lights need to be powerful enough to light the kitchen safely and clearly.* le sens du terme *light* diffère et se précise en fonction de son contexte d'énonciation. Néanmoins, le sens ne peut pas être projeté exclusivement à partir des termes d'une phrase et de leur mode de combinaison (comme dans les systèmes logiques), mais dépend de l'environnement dans lequel l'énoncé se trouve [Kempson, 1996]. En effet, lorsqu'il se trouve dans un texte long, un énoncé peut voir son sens évoluer et même parfois différer totalement des premiers choix d'interprétation [Lee et Bryant, 2002]. Par exemple, la phrase *Louis est mineur.* sera interprétée différemment, selon qu'elle soit suivie par l'une des phrases :

– Il travaille à plusieurs mètres de profondeur.
– Il ne peut conduire sans être accompagné.

Le sens d'un texte évolue au fur et à mesure qu'il est parcouru et analysé [Gayral *et al.*, 1996]. Aussi, il est important d'être en mesure de confronter l'ensemble des connaissances évoquées par un texte pour l'interpréter de manière correcte, ce qui correspond à la pragmatique.

Le contexte permet de réduire le nombre d'interprétations qu'un énoncé peut avoir. Il peut être d'ordre linguistique ou extralinguistique. Le contexte linguistique est relatif aux composants linguistiques présents et aux relations qu'ils entretiennent. Il porte sur les aspects terminologique et syntaxique du texte. Le contexte extralinguistique englobe tout ce qui est extérieur au texte et qui, pourtant, est indispensable à la compréhension de l'information véhiculée. Il porte sur les connaissances sémantiques et pragmatiques associées aux textes, utilisées pour compléter ou préciser l'information manquante ou implicite. Il est donc important de pouvoir faire le lien entre connaissances textuelles et extratextuelles. L'interprétation, pour être correcte, doit donc tenir compte de connaissances explicites, spécifiées dans le texte, et de connaissances implicites, déduites à partir du texte et de connaissances extralinguistiques. Toute la difficulté est de pouvoir représenter un ensemble de connaissances extralinguistiques suffisant au processus d'interprétation des textes LN, ainsi que d'établir le lien entre connaissances linguistiques et extralinguistiques.

Permettre à une machine d'interpréter un texte en LN suppose alors de lui donner accès à un ensemble de connaissances contextuelles suffisantes, et représentées de manière suffisamment formelles. La représentation des connaissances constitue une problématique dominante de l'IA. Dans la section suivante, nous discutons les différents for-

malismes de représentation des connaissances proposés pour aider le passage de l'informel au formel.

1.6 REPRÉSENTATION INTERMÉDIAIRE POUR LA FORMALISATION DE SPÉCIFICATIONS EN LANGAGE NATUREL

L'intérêt pour la formalisation de spécifications LN n'est pas récent [Fraser *et al.*, 1991; Vadera et Meziane, 1994], et depuis plusieurs années, divers travaux ont abordé cette problématique. Le principal verrou, réside dans la distance entre les deux formalismes, aussi la question a été abordée par l'emploi de formalismes intermédiaires permettant de 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 de connaissances formulées en langage naturel. Dans cette section, nous donnons un aperçu des formalismes pouvant jouer le rôle de pivot dans un processus de transformation de spécifications en langage naturel.

1.6.1 *Langage naturel contrôlé*

Une des solutions envisagée pour combler le fossé entre un langage naturel et un langage formel est l'utilisation d'un langage naturel contrôlé (LNC) comme intermédiaire [Schwitter, 2010]. Les LNC sont conçus à partir des langages naturels afin de réduire leur ambiguïté et leur complexité. Cette solution est souvent motivée par le fait qu'en domaine de spécialité et plus particulièrement en entreprise, l'utilisation d'un vocabulaire métier restreint et la spécification d'exigences sous des formes simplifiées sont souvent encouragées. À partir de documents rédigés dans un langage contrôlé, l'information peut être extraite à l'aide de techniques syntaxiques. Pour obtenir des performances équivalentes avec des documents en langage naturel non contrôlé, il est nécessaire d'avoir une représentation sémantique de la connaissance qui intègre des techniques de raisonnement [Mich *et al.*, 2004]. Les restrictions des LNC permettent de limiter le recours à des connaissances extérieures et à des mécanismes d'inférence.

Traditionnellement, les LNC se répartissent en deux grandes catégories selon leurs objectifs : ceux qui améliorent la lisibilité pour les lecteurs, et ceux plus formels qui permettent une analyse sémantique automatique fiable de la langue [Fuchs *et al.*, 2008]. Ces derniers ont en général une syntaxe et une sémantique fondées sur la logique, ce qui peut faciliter une mise en correspondance avec un langage formel.

Parmi les LNC les plus populaires de cette dernière catégorie, nous pouvons citer les deux langages : Attempto Controlled English (ACE) et Semantics of Business Vocabulary and Business Rules (SBVR).

ACE [Hoefler, 2004] est un langage naturel contrôlé développé à l'université de Zurich par le groupe *Attempto*. Il est à l'origine destiné à la spécification d'exigences en vue d'une traduction en Prolog [Fuchs et Schwitter, 1995]. Il est un sous-ensemble de l'anglais qui peut automatiquement et sans ambiguïté être traduit en un langage logique du premier ordre. En raison de son héritage LN, il peut être lu et compris par un locuteur anglais ; ses bases logiques permettent aux machines de le traiter. L'Attempto Parsing Engine [1] (APE) est un moteur d'analyse développé en SWI-Prolog par le groupe *Attempto*. Il permet de traduire sans ambiguïté des textes en *ACE* vers d'autres langages formels tels que OWL, SWRL ou RuleML. Une telle traduction permet aux utilisateurs de raisonner sur le texte, par exemple pour le vérifier, le valider ou y appliquer des requêtes.

SBVR [OMG, 2008] est un standard du groupe *Object Management Group* (OMG). Il est utilisé pour définir le vocabulaire et la grammaire permettant de décrire les connaissances métier de l'entreprise, pour faciliter la collaboration entre expert et ingénieur et pour assister les experts dans la spécification de règles métier d'une entreprise, en utilisant le langage naturel de façon standardisée, simple, précise et désambiguïsée.

On peut distinguer deux types d'approche faisant usage des LNC formels. Premièrement les approches qui prennent le parti de les utiliser comme point de départ d'une traduction automatique vers un modèle plus formel [Smith *et al.*, 2002; Gervasi et Zowghi, 2005; Gordon et Harel, 2009]. Deuxièmement, celles qui prennent comme point de départ les textes en LN et s'attaquent à leur transformation en LNC afin de réduire l'écart formel [Bajwa *et al.*, 2010; Njonko et El Abed, 2012; Guissé *et al.*, 2012] dans l'optique d'effectuer un passage vers le formel en deux temps. Le travail de Guissé [2013] s'inscrit dans cette logique et constitue une première phase dans le processus de formalisation automatique de règles métier spécifiées en LN. Pour ce faire il propose de transformer des règles métiers issues de textes réglementaires en LN vers des règles SBVR [Guissé *et al.*, 2012]. De même [Njonko et El Abed, 2012] utilisent SBVR comme représentation intermédiaire pour la transformation de règles métiers spécifiées en LN vers des modèles exécutables tels que UML ou SQL. Enfin, Bajwa *et al.* [2012] proposent l'outil NL2Alloy pour générer à partir de contraintes en LN du code Alloy [2]. NL2Alloy utilise SBVR comme représenta-

1. http ://attempto.ifi.uzh.ch/site/

2. Un langage et outil pour la vérification de modèle relationnelle. http ://alloy.mit.edu/alloy/

tion pivot. La méthode de transformation utilisée met en œuvre une chaîne de transformations exploitant les trois outils NL2SBVR [Bajwa *et al.*, 2011], SBVR2OCL [Bajwa et Lee, 2011] puis UML2Alloy [Anastasakis *et al.*, 2007]. La première transformation consiste à passer de contraintes en langage naturel vers des règles SBVR puis de règles SBVR vers des contraintes UML/OCL et enfin de passer des contraintes OCL à du code Alloy.

Une des limitations des langages naturels contrôlés et plus particulièrement de SBVR est qu'ils manquent souvent d'outils d'édition, de mécanismes de validation et de moteurs d'inférence [Sukys *et al.*, 2011]. 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 [Ceravolo *et al.*, 2007; Karpovic et Nemuraite, 2011; Sukys *et al.*, 2012].

1.6.2 *Modèle de représentation des connaissances*

La modélisation d'un système a pour but de donner une vision abstraite et simplifiée du système à réaliser. Différents langages de modélisation sont proposés à cet effet. Parmi les plus utilisés, nous pouvons citer UML (et ses dérivés SysML, MSC, LSC), RDF ou OWL. Ces modèles ont une place non négligeable dans le processus de spécification d'exigences et ont comme objectif d'aider à spécifier, visualiser, documenter et construire des modèles de systèmes. Ils sont souvent orientés objet et peuvent proposer une notation graphique pour faciliter la compréhension lors des premières phases du processus de développement.

L'objectif du processus de spécification d'exigences est d'obtenir un ensemble de spécifications précises, complètes et communes aux parties prenantes. Cependant, ce processus débute le plus souvent par des spécifications informelles, issues de connaissances hétérogènes et propres à chacune des parties prenantes. Les modèles de représentation des connaissances permettent d'organiser et de représenter le savoir humain, de sorte à faciliter l'accès aux connaissances et leur compréhension. L'usage de ces modèles dans un processus de développement de système est à l'étude depuis plusieurs années [Lin *et al.*, 1996; Charnois, 1999; Kaiya et Saeki, 2006; Körner et Brumm, 2010; Farfeleder *et al.*, 2011; Landhäußer *et al.*, 2013]. On peut distinguer deux types de modèles. Les modèles semi-formels et les modèles formels. Ces deux modèles se côtoient souvent dans un même projet, car ils sont complémentaires et pallient chacun les inconvénients de l'autre. Les modèles semi-formels offrent une représentation synthétique et intuitive. Ils sont connus pour être aisément compréhensibles

par un humain. Néanmoins, ces modèles peuvent être ambigus sur certains points, car leur sémantique n'est pas complètement définie. La construction de systèmes sur la base de ces modèles de représentation peut alors conduire à des modèles ambigus. Les modèles formels fournissent quant à eux une représentation précise et non ambiguë, et un formalisme qui autorise l'application de méthodes de vérification formelles. Cependant, leur usage demande souvent une expertise plus poussée.

1.6.2.1 *Modèle de représentation semi-formel*

Les modèles de représentation semi-formels les plus connus sont UML [UML] et ses dérivés tels que *SysML*, *MSC* ou *LSC*. Le langage UML est un standard de l'*Object Management Group* (OMG), largement accepté et utilisé pour modéliser des systèmes logiciels. Il s'appuie sur une approche orientée objet, et propose un ensemble de diagrammes permettant de modéliser la structure et le comportement d'un système. UML propose plusieurs types de diagrammes pour modéliser différents aspects d'un système. Il est considéré comme un langage semi-formel car si sa syntaxe est bien définie, sa sémantique elle, ne l'est que partiellement. Le Systems Modeling Language (SysML) est un langage de modélisation graphique dérivé d'UML et spécifiques aux systèmes. Il est plus réduit et moins restrictif que UML qui est centré sur les systèmes logiciels. Le Message Sequence Charts (MSC) [MSC] est un formalisme visuel permettant de décrire des diagrammes d'interaction visant à représenter d'une manière très intuitive et transparente le comportement de communication des composants du système et de leur environnement au moyen d'échange de messages. MSC ne fournit pas les moyens de distinguer entre un comportement obligatoire et possible. Le Live Sequence Chart LSC [Damm et Harel, 2001] étend la sémantique de MSC pour pouvoir spécifier ce type de comportement [Brill *et al.*, 2004]. Il introduit la distinction entre obligatoire et possible au niveau des diagrammes et fournit des moyens pour spécifier le temps d'activation souhaité par un état ou par une séquence de communication.

L'élaboration de modèles semi-formels à partir de spécifications d'exigences peut être longue et fastidieuse. Aussi, un grand nombre de travaux se sont intéressés à l'automatisation de ce processus afin d'extraire à partir de texte en langage naturel un modèle UML [Ilieva et Boley, 2008; Tichy et Koerner, 2010; Bajwa *et al.*, 2010; Landhäußer *et al.*, 2013] ou MSC [Kof, 2009]. Comme les spécifications LN peuvent toujours contenir des ambiguïtés, des interactions manuelles avec le concepteur ne peuvent en général être exclues, conduisant à des approches (semi-) automatiques [Drechsler *et al.*, 2012]. Les

approches proposées font le plus souvent usage de techniques du traitement automatique des langues (TAL). Elles s'avèrent particulièrement concluantes lorsqu'un dialogue interactif avec le concepteur est employé [Keszocze *et al.*, 2013].

Tichy et Koerner [Tichy et Koerner, 2010] font appel au TAL et aux technologies sémantiques pour générer des modèles UML à partir d'exigences LN. Les spécifications sont d'abord améliorées à l'aide de l'outil RESI [Körner et Brumm, 2009] fondé sur l'utilisation d'ontologies du domaine. Par la suite, une collection de 60 rôles thématiques et de règles de transformations associées génèrent un modèle UML directement à partir des documents LN. Kof [2009] analyse des scénarios textuels à l'aide de techniques de TAL afin de générer un diagramme MSC. Il se sert de ce passage pour identifier les informations absentes des scénarios textuels, afin de combler les informations manquantes. Ensuite, l'analyste des besoins valide le MSC généré.

Le manque d'une sémantique formelle dont souffre UML peut mener à de sérieux problèmes de modélisation [Chama *et al.*, 2013], générateurs d'incohérences dans les modèles développés [Ledang et Souquières, 2001]. De plus, sa simplicité a comme prix, un manque de précision [Giese et Heldal, 2004]. Ces points faibles ont motivé plusieurs travaux sur le passage d'UML vers des langages formels tels que B [3] [NGUYEN, 1998; Ledang et Souquières, 2001; Laleau et Mammar, 2001], vers Alloy [Anastasakis *et al.*, 2007], vers CSP [4], Object-Z [5] et différents types de réseaux de Petri [Gulan *et al.*, 2013] ou encore vers Maude [Gagnon *et al.*, 2008; Mokhati et Badri, 2009; Chama *et al.*, 2013].

La notation UML est largement fondée sur les diagrammes. Cependant, les diagrammes ne fournissent pas le niveau de concision et d'expressivité qu'un langage textuel peut offrir [Clavel et Egea, 2006b]. UML peut être complété par le langage d'expression de contraintes OCL [6] [OCL]. L'*Object Constraint Language* (OCL) permet de compléter les modèles UML par la définition de contraintes qui ne peuvent être exprimées graphiquement, et que le modèle doit satisfaire. Il permet de spécifier des invariants et des pré-et post-conditions à vérifier [Roldan et Duran, 2010]. Divers travaux ont proposé la validation de contraintes OCL par une transformation en spécifications formelles, notamment vers le langage Maude [Clavel et Egea, 2006b; Roldan et Duran, 2010; Duràn et Roldàn, 2012].

3. B est une méthode formelle de développement logiciel qui couvre le processus logiciel à partir de la spécification abstraite jusqu'à l'application exécutable. Il permet de modéliser de façon abstraite le comportement d'un programme.

4. CSP, est un langage de description de modèles d'interaction. Il correspond à une une algèbre de processus permettant de modéliser l'interaction de systèmes.

5. http ://itee.uq.edu.au/~smith/objectz.html

6. OCL est standard de l'Object Management Group (OMG)

À notre connaissance, il existe peu d'approches proposant de prendre une représentation UML comme pivot d'un processus complet de passage du langage naturel vers des spécifications formelles. Seul l'outil NL2Alloy [Bajwa *et al.*, 2012] (décrit à la section précédente) propose une telle utilisation. NL2Alloy combine une succession d'outils permettant de passer de contraintes rédigées en langage naturel à des règles SBVR (NL2SBVR), puis vers de l'UML/OCL (SBVR2OCL), pour enfin aboutir au langage Alloy mais cela sans vérifications des représentations intermédiaires.

1.6.2.2 *Modèle de représentation formel*

Les modèles de représentation formels sont du type réseau sémantique, tels que les graphes conceptuels et les ontologies. Leur fondement mathématique autorise divers mécanismes de raisonnement. Fougères et Trigano [1997] prennent les graphes conceptuels comme pivot entre spécifications LN et spécifications formelles en Z.

Le recours aux ontologies permet d'améliorer considérablement la qualité des exigences spécifiées [Omoronyia *et al.*, 2010], car elles permettent une compréhension commune et une communication entre des personnes ayant des besoins et des points de vue provenant de contextes différents [Uschold et Gruninger, 1996]. Elles permettent d'expliciter et de formaliser la terminologie d'un domaine, en donnant un sens unique aux concepts et propriétés représentant le vocabulaire conceptuel. Chaque élément du vocabulaire possède alors une interprétation unique, qui est partagée par l'ensemble des utilisateurs. Ce cadre unificateur permet de passer des jargons personnels à une terminologie commune, ce qui peut atténuer les malentendus au sujet de termes et permettre de réduire la confusion terminologique et conceptuelle.

Les contraintes et restrictions d'un domaine sont connues par les experts du domaine. L'utilisateur, n'étant pas familier du domaine et, en particulier de ses contraintes, peut être amené à spécifier des exigences en contradiction avec celles-ci. De nombreuses méthodes fondées sur l'utilisation d'une ontologie ont été proposées avec comme but l'obtention d'exigences correctes et non ambiguës [Kaiya et Saeki, 2006; Lee et Zhao, 2006; Shibaoka *et al.*, 2007; Körner et Brumm, 2010; Al Balushi *et al.*, 2013].

L'utilisation des ontologies comme moyen de représenter les connaissances est de plus en plus acceptée dans différents domaines. Leur potentiel pour améliorer la compréhension commune, préciser et organiser les connaissances, amène une quantité croissante de travaux à s'intéresser à leur utilisation dans le domaine de l'ingénierie des exigences (IE). Lin *et al.* [1996] ont proposé l'une des premières

approches de représentation d'exigences à l'aide d'une ontologie implémentée en utilisant *Prolog*.

Depuis, plusieurs chercheurs se sont penchés sur les avantages que pourrait avoir l'introduction d'ontologies, pour la spécification [Souag., 2012], l'interopérabilité [Boukhari *et al.*, 2012], la validation [Körner et Brumm, 2009; Chniti *et al.*, 2012] ou l'amélioration [Körner et Brumm, 2009; Castañeda *et al.*, 2012] d'exigences. De plus en plus de travaux insistent sur la pertinence de l'usage d'ontologies dans le processus d'ingénierie des exigences [Körner et Brumm, 2009; Farfeleder *et al.*, 2011]. Pourtant, leur utilisation comme représentation intermédiaire dans un processus de formalisation automatique de spécifications LN n'a pas encore été explorée, même si, différentes approches et outils ont été proposés pour la vérification et l'amélioration de spécifications LN [Körner et Brumm, 2009; Bajwa *et al.*, 2011; Castañeda *et al.*, 2012]. L'outil RESI [Körner et Brumm, 2009] a pour objectif d'améliorer la qualité de spécifications d'exigence, en aidant l'analyste à identifier certains problèmes de spécifications et à prendre des décisions. RESI s'appuie sur une ontologie telle que Cyc, ConceptNet ou Wordnet, que les auteurs définissent comme représentant "le bon sens". L'outil identifie les points critiques, mais ne remplace pas le décideur humain, sauf dans des cas précis. Il peut par exemple reconnaître des informations spécifiées de manière incomplète et dans ce cas demande à l'utilisateur de donner l'information manquante. Il permet surtout de vérifier la cohérence du modèle produit à partir des spécifications. Dans notre contexte, il est important d'aller plus loin, non seulement reconnaître les éléments sous-spécifiés ou implicites mais aussi inférer l'information manquante.

L'utilisation d'une représentation intermédiaire semble donc être la meilleure alternative. Cette représentation intermédiaire doit jouer le rôle de pivot pour faciliter le passage de spécifications LN en spécifications formelles, en réduisant la distance formelle entre elles.

1.7 DE L'INFORMEL AU FORMEL

La Figure 1 (inspirée de [Uschold et Gruninger, 2004]) a pour objectif, de résumer les différents niveaux de formalisation des langages et modèles de représentation discutés au travers de l'état de l'art. Elle définit un continuum dans lequel, plus on va vers la droite, plus le degré de formalisation augmente, réduisant ainsi l'ambiguïté jusqu'à permettre la possibilité d'un raisonnement formel automatisé. À l'extrémité gauche se trouve le langage naturel qui, nous l'avons vu est considéré comme non formel. Il correspond au langage le plus expressif, mais son manque de précision rend difficile l'application d'un raisonnement automatisé sur les connaissances qu'il véhicule. À

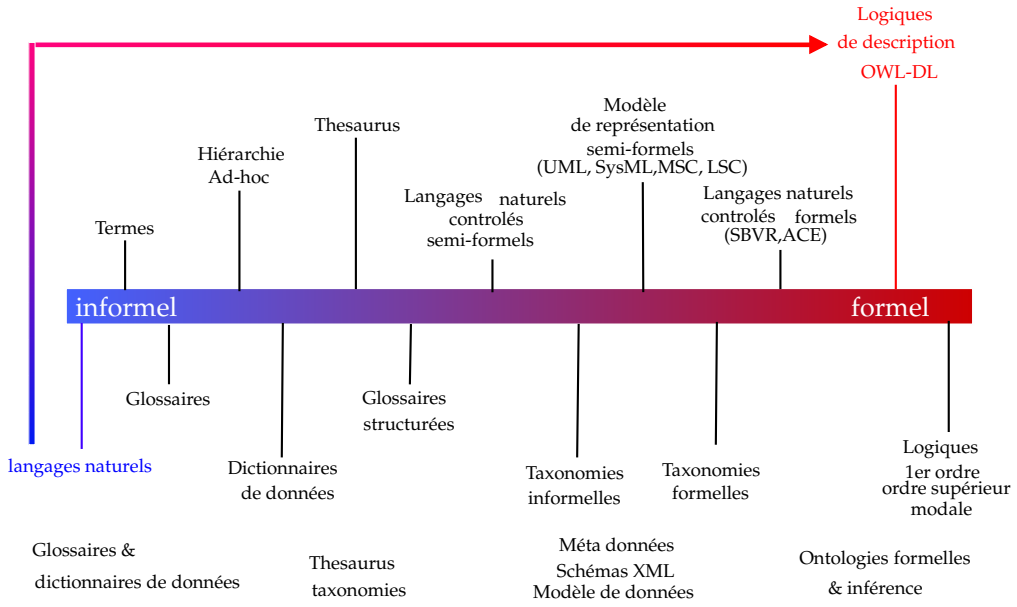


FIGURE 1: Niveaux de formalisation des langages de représentation des connaissances

l'extrémité droite, se trouvent les logiques du premier ordre, d'ordre supérieur ou modale. Ces logiques sont très expressives, mais ne permettent pas toujours un raisonnement décidable [Guarino *et al.*, 2009]. Dans notre contexte, ce que l'on recherche est un compromis entre expressivité, pour la représentation des connaissances formulées en langage naturel et possibilité de raisonnement automatisé, pour la vérification formelle des connaissances représentées. Les logiques de description à l'aide desquelles sont traditionnellement représentées les ontologies sont des sous-ensembles moins expressifs de la logique du premier ordre, qui disposent de raisonneurs décidables et efficaces. C'est donc vers l'utilisation de ces formalismes et en particulier l'utilisation d'ontologies formelles que nous nous sommes dirigés pour articuler le passage entre informel et formel.

1.8 NOTRE APPROCHE

Notre ambition est de permettre à un utilisateur de configurer le comportement d'un système à l'aide de descriptions écrites en langage naturel [Sadoun *et al.*, 2011]. À cette fin l'utilisateur est invité à spécifier ses besoins en fonction d'une *description technique* du système à réaliser. Nous qualifions ici de *description technique* un ensemble de composants du système ainsi que leurs caractéristiques. Ces caractéristiques peuvent découler de choix relatifs au matériel (type de composants, nombre, capacité), aux dispositions ou agencements des différents composants ou encore aux interactions possibles entre com-

posants. Les spécifications utilisateur doivent alors tenir compte de la description technique du système pour décrire ses règles de comportement. L'utilisateur visé n'étant pas supposé rompu à l'art de spécifier des exigences, les spécifications qu'il énonce ont de fortes chances d'être ambiguës, incomplètes ou incohérentes. Aussi, la vérification des exigences spécifiées s'impose.

1.8.1 *Schéma général*

L'approche de formalisation et de vérification des exigences utilisateurs que nous proposons peut se résumer par les étapes ci-dessous, illustrées sur la figure 2 :

1. Modéliser manuellement l'ontologie qui représente le cadre générique du comportement du système, à partir du domaine de connaissance et de la description technique du système.
2. Analyser les spécifications d'exigence en LN, guidé par l'ontologie.
3. Peupler (spécialiser) l'ontologie à partir de l'analyse des spécifications.
4. Traduire l'ontologie spécifique aux besoins utilisateur en spécifications formelles. Le langage choisi a été Maude.
5. Vérifier les modèles issus des spécifications LN.
6. Le résultat des vérifications conditionne la suite du processus. Si le modèle s'avère correct, alors la phase de réalisation du système peut être lancée. Sinon les incohérences et informations manquantes détectées sont soumises à l'utilisateur afin qu'il puisse corriger et améliorer la spécification de ses exigences. L'utilisateur interagit ainsi avec le modèle jusqu'à ce qu'il soit cohérent et conforme à ses besoins.

Dans la suite, nous décrivons les règles de comportement que nous souhaitons identifier à partir des spécifications d'exigences ainsi que les vérifications qui doivent être appliquées. Ensuite, nous décrivons le modèle de représentation choisi, puis les méthodes d'instanciation et de vérification appliquées à ce modèle.

1.8.2 *Définition des règles de comportement*

Les règles que nous souhaitons identifier à partir des spécifications sont des règles logiques. Chaque règle est constituée d'un antécédent et d'un conséquent. Antécédent et conséquent sont eux-mêmes constitués de prédicats. Un prédicat est une propriété, ou proposition pou-

vant porter sur une ou plusieurs entités du domaine. Il s'agit alors de reconnaître les prédicats de ces deux parties dans les textes.

1.8.3 *Définition des vérifications souhaitées*

Afin de garantir la qualité et la conformité des règles de comportement issues des exigences, celles-ci doivent être vérifiées conformément aux critères détaillés ci-dessous :

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, i.e. vérifier que les prédicats peuvent être instanciés au sein de l'ontologie. Dans le cas de la définition d'un comportement du système, cela dépend en grande partie de la description technique du système qui n'est pas déterminée par l'utilisateur mais qui lui est imposée. Cette description définit les interactions possibles entre les composants du système. L'utilisateur peut par exemple, souhaiter une interaction entre deux composants du système qui, dans la pratique ne peuvent interagir i.e. qui ne sont pas liés au sein de l'ontologie. Dans ce cas l'utilisateur doit être averti afin de réviser ses exigences ou veiller à ce que la configuration physique du système évolue.

COHÉRENCE DE LA RÈGLE Une fois une règle avérée bien constituée, sa cohérence doit être vérifiée. La cohérence d'une règle par rapport à l'ensemble des règles est établie relativement à son conséquent. 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. Dans le second cas, les antécédents sont différents mais peuvent s'appliquer en même temps. L'identification de ce cas n'est pas triviale et nécessite une vérification de l'ensemble des états du système.

CONFORMITÉ DES RÈGLES Les règles sont dites conformes si le comportement qu'elles décrivent répond à des exigences spécifiées soit par l'utilisateur ou par le concepteur du système. Par exemple, l'ensemble de règles doit permettre qu'à partir d'une configuration donnée, tous les états du système devant être atteints le soient et qu'aucun état indésirable ne soit jamais atteint.

Afin de pouvoir effectuer ces vérifications, les spécifications d'exigences doivent être transformées en spécifications formelles. Du fait du fossé entre spécifications LN et spécifications formelles, une trans-

formation directe n'est pas envisageable. Nous proposons donc de passer par l'intermédiaire d'une représentation pivot afin de réduire l'écart entre spécifications LN et formelles. La transformation s'effectue alors en deux temps : d'abord entre les spécifications LN et un modèle intermédiaire puis, entre le modèle intermédiaire et un langage de spécification formelle.

La représentation intermédiaire a été conçue pour permettre une formalisation des connaissances décrites en langage naturel tout en conservant la sémantique des exigences formulées. Notre choix de modèle s'est orienté vers une ontologie formelle. Quant au langage de spécification formelle, nous avons opté pour Maude, en cela qu'il est compatible et complémentaire de OWL.

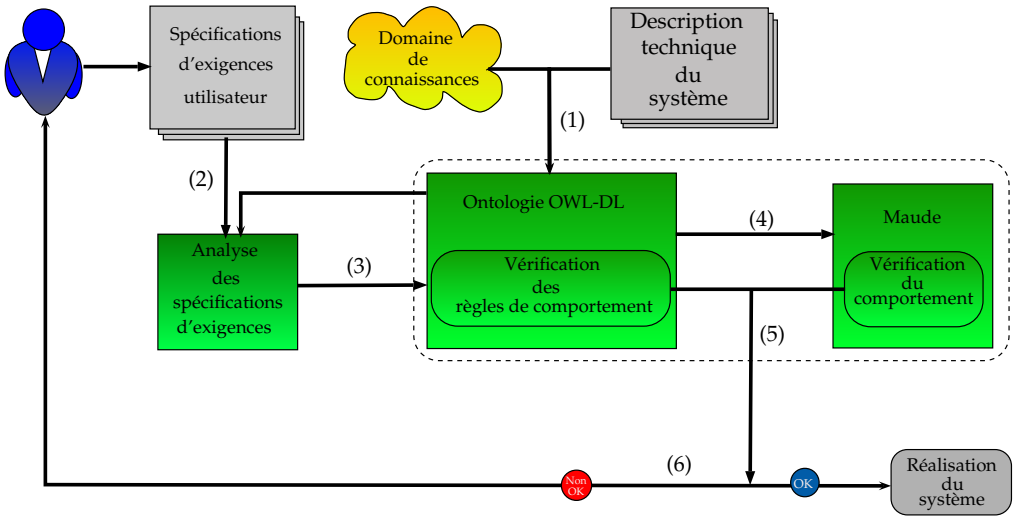


FIGURE 2: Architecture du processus de configuration

1.8.4 *Le modèle*

Nous entendons par ontologie formelle, une ontologie qui, en plus de représenter les connaissances d'un domaine, s'appuie sur une logique de description et donc permet de faire des inférences. Ce choix est motivé par différents facteurs. Tout d'abord, une ontologie a l'avantage d'être compréhensible par un humain et interprétable par une machine car elle est définie sur un vocabulaire de termes en langage naturel dont elle explicite la signification de manière formelle. De plus, ces dernières années plusieurs travaux se sont intéressés à la construction automatique d'ontologies à partir de textes en LN et d'autre part à l'utilisation d'ontologie pour guider l'extraction d'information. Les résultats obtenus confirment la possibilité d'un passage automatique des textes aux ontologies et le rôle important que celles-

ci peuvent jouer dans l'acquisition de connaissances à partir de textes en LN. En outre, les mécanismes d'inférence associés aux ontologies permettent non seulement de déduire de nouvelles connaissances, mais aussi de vérifier la cohérence des connaissances modélisées [Sadoun *et al.*, 2012]. Le formalisme logique des ontologies autorise une transformation simple et intuitive vers un langage de spécification formel. Enfin, l'ontologie, en tant que modèle de représentation conçu pour capturer la sémantique du langage naturel, permet de garder une trace du lien entre LN et spécifications formelles et donc de faciliter la navigation entre spécifications LN et spécifications formelles.

Différents langages d'ontologie ont été proposés. Ils sont en général fondés sur les logiques du premier ordre ou plus particulièrement sur les logiques de description. Les logiques du premier ordre permettent une expressivité élevée sans garantir la décidabilité. Les logiques de description permettent de représenter les connaissances d'un domaine de manière formelle et structurée. Elles sont des fragments des logiques du premier ordre et offrent un bon compromis entre expressivité et décidabilité. Nous avons choisi le langage OWL-DL, une version décidable de OWL fondée sur les logiques de description pour définir l'ontologie qui fera le lien entre spécifications LN et formelles. OWL est l'un des langages d'ontologie les plus populaires, considéré comme le standard actuel du web sémantique. OWL autorise la création d'autant de propriétés et d'annotations que nécessaires à une représentation efficace et compréhensible des connaissances issues des textes, ce qui peut permettre dans le contexte d'un peuplement automatique d'ontologie, de maintenir le lien entre les éléments du texte et leur représentation sémantique. Des unités linguistiques peuvent ainsi être liées aux instances de l'ontologie auxquelles elles réfèrent.

Dans les textes, la référence à une instance de concept se fait par son nom (*Aristote*) ou par la mention des propriétés qu'elle possède (*disciple de Platon* et *précepteur d'Alexandre le Grand*). Dans ce dernier cas, l'être humain fonde son raisonnement sur ces propriétés pour inférer le type de l'instance de concept (sa classe sémantique) ou l'identifier parmi d'autres instances. Les propriétés en cause définissent alors des conditions d'identification. Nous distinguons quatre types de propriétés *définissantes* :

1. propriétés nécessaires : pour appartenir à un concept, un individu doit obligatoirement posséder ces propriétés nécessaires. Par exemple, un individu du concept *Voiture* doit nécessairement avoir des roues, un individu du concept *Oiseau* doit nécessairement avoir des ailes. À l'inverse, un individu ayant des ailes n'est pas nécessairement un oiseau.

2. propriétés suffisantes : les individus possédant ces propriétés sont reconnus comme étant membres d'un concept. Par exemple, un individu possédant la propriété *enceinte* peut être inféré comme appartenant au concept *Femme*. Cependant, une femme ne doit pas nécessairement posséder la propriété *enceinte*.
3. propriétés nécessaires et suffisantes : elles définissent une équivalence sémantique entre un concept et certaines de ses propriétés.
4. propriétés d'unicité : un individu qui possède ces propriétés peut être identifié de façon unique par rapport aux autres individus. Par exemple, le *numéro d'inscription* d'un *étudiant* permet de l'identifier de façon unique.

Le troisième type de propriété qui subsume les deux premiers permet la classification des individus. Le dernier type lui, permet la distinction des individus.

Lors de la définition d'un concept, il faut donc choisir les propriétés *essentielles* qui participent à sa définition [Bouaud *et al.*, 1994]. De notre point de vue, une ontologie formelle doit nécessairement définir pour chacun de ses concepts, les conditions de classification et d'identification de ses individus, de sorte à permettre l'identification univoque de chaque individu de l'ontologie et ainsi lever toutes les ambiguïtés [Sadoun *et al.*, 2013b]. Par la suite, nous montrons de quelle manière l'exploitation de propriétés définissantes peut guider et faciliter le processus d'instanciation du modèle construit.

Le modèle ontologique que nous proposons représente un cadre sémantique générique permettant de guider un processus de peuplement d'ontologie à partir de spécifications LN, d'inférer les connaissances implicites dans les textes et de vérifier la cohérence de l'ensemble des connaissances modélisées. En outre, le formalisme logique de ce modèle autorise une transformation des connaissances issues des spécifications LN en spécifications formelles.

1.8.5 *Instanciation du modèle*

Notre approche consiste à modéliser le comportement générique du système à réaliser, puis à spécialiser ce modèle en fonction des spécifications utilisateur [Sadoun *et al.*, 2013a]. Nous montrons de quelle façon une ontologie formelle représentant le comportement générique d'un système peut constituer un cadre formel pouvant faciliter l'interprétation et la vérification automatique de spécifications d'exigences utilisateur. Dans notre contexte, la spécialisation de l'ontologie consiste à y ajouter un ensemble d'instances de concepts (individus) et de propriétés identifiées à partir de leurs mentions dans les

textes ainsi que les instances de règles de comportement reconnues. Ce processus correspond au peuplement de l'ontologie.

Afin d'extraire ces connaissances des textes, nous proposons de guider l'analyse des spécifications d'exigences par l'ontologie. Les informations issues des textes sont associées aux connaissances modélisées dans l'ontologie. Leur interprétation se fait à l'aide des connaissances pragmatiques modélisées dans l'ontologie et non à partir des connaissances du texte seul. Comme les individus sont identifiables par leurs propriétés sémantiques, nous proposons de guider le peuplement de l'ontologie par l'identification d'instances de propriétés [Sadoun, 2012] et ainsi d'inférer les classes sémantiques des individus au sein de l'ontologie, plutôt qu'au niveau du texte [Sadoun *et al.*, 2013b]. Les instances de propriété correspondent à des triplets (propriété, domaine, image) ; aussi dans les textes, nous cherchons à identifier des triplets de termes dénotant des instances de propriétés. L'identification de triplets dénotant des instances de propriétés de l'ontologie se fait à l'aide de règles acquises à partir des formes lexico-syntaxiques récurrentes et de connaissances ontologiques.

Les règles de comportement que nous souhaitons reconnaître correspondent à des règles logiques constituées de prédicats. La reconnaissance d'une règle passe alors par l'identification des prédicats qui la composent. L'identification de règles de comportement spécifiques aux besoins est guidée par un ensemble de règles de comportement génériques explicitement décrites dans l'ontologie. L'ontologie comporte deux sortes de prédicats, les prédicats unaires, i.e. les concepts, et les prédicats binaires, i.e. les propriétés. Ces prédicats portent sur les individus (instances de concepts), sur des valeurs de types simples (entier, chaîne de caractères, etc.) définies dans OWL ou enfin sur des variables. Les prédicats à identifier à partir des textes ainsi que leurs arguments doivent donc appartenir au vocabulaire défini par l'ontologie. Plus précisément, dans notre approche, les prédicats à identifier devront dénoter des instances de propriété.

Guider l'analyse des spécifications par l'ontologie permet de reconnaître dans les spécifications des règles utilisateurs, de vérifier leur constitution, de contrôler l'appartenance des individus sur lesquels portent leurs prédicats au vocabulaire conceptuel de l'ontologie, ainsi que de vérifier la cohérence de chaque prédicat par rapport au modèle existant, et cela grâce au raisonnement ontologique.

1.8.6 *Vérifications possibles*

La première étape de vérification concerne la *cohérence de la règle* par rapport au modèle ontologique. Le cas le plus simple à vérifier est l'existence d'une autre règle ayant le même antécédent mais un

conséquent contradictoire. OWL permet de définir les propriétés en contradiction, i.e. les propriétés qu'un individu ne peut posséder simultanément. Lors de l'identification des prédicats d'une nouvelle règle nous recherchons alors la présence d'une règle contenant un conséquent contradictoire. Pour ce faire, nous faisons appel au langage d'interrogation d'ontologie SQWRL. Ce langage de requête fournit des opérations de type SQL pour extraire des connaissances à partir de OWL. Il se peut également que deux règles ne contenant pas les mêmes antécédents amènent à des conséquents contradictoires. Ce cas d'incohérence est moins aisé à détecter. En effet, ceci nécessite de pouvoir explorer les différentes possibilités d'exécution des règles. Comme cela ne peut se faire sous OWL, nous proposons soit de soumettre les antécédents des règles à l'utilisateur qui pourra déterminer s'ils peuvent se produire simultanément dans sa configuration ou de vérifier ce cas particulier, en utilisant les méthodes formelles possibles de par la traduction du modèle intermédiaire obtenu en une spécification formelle Maude.

Le troisième type de vérification concerne l'*application de la règle*. L'application d'une règle dépend du déclenchement de son antécédent. Ce dernier dépend à son tour de la satisfaction de ses prédicats. La vérification consiste alors à contrôler à l'aide de requêtes SQWRL, l'existence d'instances de concepts ou de propriétés pouvant correspondre à chacun des prédicats binaires de l'antécédent. Si aucune instance n'existe pour un prédicat, alors la règle est inapplicable et s'identifie à du *code mort*.

Le dernier type de vérification qui nous intéresse porte sur *la conformité des règles*. Le raisonnement permis par OWL est monotone, c'est à dire que l'ajout de nouvelles connaissances ne peut modifier les connaissances initiales. OWL ne permet donc pas de représenter l'évolution des états du système. D'autre part, le raisonnement OWL s'effectue sur l'ensemble du modèle ontologique et ne peut s'effectuer sur un séquencement particulier de règles. Aussi, l'atteignabilité d'une configuration particulière ne peut se vérifier sous OWL. Elle peut cependant s'effectuer sous Maude, car ce langage permet une exploration des différentes interprétations possibles d'un modèle, i.e. des différentes configurations possibles du système.

L'utilisation d'une ontologie permet donc d'effectuer des vérifications sur les règles utilisateur tout au long du processus de peuplement, permettant ainsi d'identifier les erreurs au plus tôt. Son formalisme logique autorise de surcroît une traduction vers un langage de spécifications formelles pouvant réaliser les vérifications manquantes et ainsi compléter les vérifications à effectuer.

1.9 CONCLUSION

Le passage de spécifications en langage naturel à des spécifications formelles vérifiables suscite un intérêt de plus en plus grand. L'automatisation de ce processus demeure une tâche difficile. Dans la littérature, il est entendu qu'un passage direct de spécifications informelles à des spécifications formelles n'est pas envisageable. La solution majoritairement proposée et acceptée par l'état de l'art est le passage par une représentation intermédiaire pivot qui permettrait de réduire l'écart entre les deux types de spécifications. Dans cette section, nous avons énoncé les critères que doit satisfaire une telle représentation.

L'expressivité et les mécanismes de raisonnement offert par une ontologie, en font un bon candidat pour jouer le rôle de pivot dans un processus de passage automatisé de l'informel au formel. Son usage pour cette tâche a pourtant été peu exploré. Notre approche de formalisation de spécifications en langage naturel se fonde sur l'exploitation du potentiel de représentation et de formalisation d'une ontologie en OWL DL et des raisonneurs associés pour interpréter des spécifications LN.

2

ONTOLOGIE : MODÉLISATION ET PEUPLEMENT

Sommaire

« La connaissance des mots conduit à la connaissance des choses. »

Platon

Introduction

Dans ce chapitre, nous donnons une définition formelle des ontologies et introduisons les notations que nous utiliserons tout au long du manuscrit. En section 2.2, nous décrivons le développement d'une ontologie, pour lequel nous distinguons deux étapes : la conceptualisation et le peuplement. Après avoir décrit la conceptualisation d'une ontologie dans ses grandes lignes et mis en avant les enjeux du processus de conceptualisation, nous donnons en section 2.3 une description détaillée de l'état de l'art du peuplement, vis-à-vis duquel nous nous positionnons par rapport à différents critères.

2.1 ONTOLOGIE

2.1.1 *Définition*

À l'origine, la notion d'ontologie est issue de la philosophie qui la considère comme une branche de la métaphysique s'intéressant aux propriétés qui définissent l'existant. Elle est, d'après *Aristote*, l'*étude de l'être en tant qu'être*. Pour l'informatique, l'ontologie est en premier lieu un modèle de représentation des connaissances d'un domaine. Aussi, construire une ontologie consiste à décider de la manière d'être et d'exister des objets d'un domaine [Charlet *et al.*, 2004], c'est-à-dire de décider à quels concepts ils appartiennent et quelles propriétés ils possèdent. Selon Swartout *et al.* [1997], une ontologie correspond à un ensemble de termes hiérarchiquement structurés pour décrire un domaine et pouvant être utilisée comme le squelette d'une base de connaissances. L'ontologie définit les concepts et propriétés de base constituant le vocabulaire d'un domaine spécifique, ainsi que les règles permettant de combiner les concepts et les propriétés afin de définir des extensions du vocabulaire [Neches *et al.*, 1991]. Gruber [1995] définit une ontologie comme *une spécification formelle et explicite d'une conceptualisation partagée*. Formelle, car elle spécifie de manière précise et sans ambiguïtés les connaissances du domaine. Explicite, car elle fournit les moyens de décrire explicitement la conceptualisation des connaissances représentées [Bernaras *et al.*, 1996]. Partagée, car elle correspond à un modèle partagé d'un domaine qui encode une vue commune à différentes parties [Bouquet *et al.*, 2003].

Pour résumer, l'ontologie est un consensus accepté par une communauté [1] ayant pour but de représenter une compréhension commune d'un domaine. Elle définit les objets du monde, leurs propriétés et leurs contraintes de manière explicite, définissant ainsi un vocabulaire précis du domaine qu'elle organise et formalise afin de le rendre interprétable tant par les humains que par les machines. Elle fournit une sémantique formelle aux connaissances qu'elle explicite de sorte à permettre l'application de raisonnement. L'utilisation d'une ontologie a donc essentiellement deux objectifs :

1. représenter les connaissances d'un domaine ;
2. permettre de raisonner sur ces connaissances.

2.1.2 *Définition formelle et notations*

Cette section a pour vocation de définir une ontologie de manière formelle, d'introduire le vocabulaire et de poser les différentes notations que nous emploierons par la suite.

Une ontologie définit les concepts ($\mathbb{C}$), les propriétés ($\mathbb{P}$) et les individus ($\mathbb{I}$) d'un domaine, tels que $\mathbb{C}$, $\mathbb{P}$ et $\mathbb{I}$ sont trois ensembles disjoints. Les concepts et propriétés d'une ontologie sont définis à l'aide d'axiomes terminologiques ($\mathbb{A}$). Les axiomes terminologiques sont une collection de formules typiquement décrites en logique de description portant sur les concepts et propriétés. Ils définissent les connaissances fondamentales de l'ontologie supposées vraies au cours d'un raisonnement.

DEFINITION 1: Un **concept** C ($C \in \mathbb{C}$) définit un ensemble d'individus ayant une sémantique et des propriétés communes. Il peut lui-même se décliner en sous-concepts.

DEFINITION 2: Une **propriété** P ($P \in \mathbb{P}$) permet de définir des relations entre individus ou des couples attributs/valeurs. Elle est définie entre un domaine noté $\mathbb{D}$ (qui est un sous-ensemble de $\mathbb{C}$) et une image notée $\mathbb{R}$ [2] (qui est l'union d'un sous-ensemble de $\mathbb{C}$ et d'un ensemble de types simples). Dans la suite, la notation $P.C_R$ ($P \in \mathbb{P}$ et $C_R \in \mathbb{R}$) signifie que les valeurs d'image de la propriété P appartiennent à la classe sémantique C_R.

DEFINITION 3: Les **axiomes terminologiques** ($\mathbb{A}$) permettent de définir les relations qu'entretiennent concepts et propriétés. Ces axiomes définissent trois types de relations :

1. Guarino [1997] insiste sur le fait qu'une ontologie représente un accord possiblement incomplet sur une conceptualisation.
2. $\mathbb{R}$ fait référence à *range* qui est la notation OWL pour l'image d'une propriété.

1. les **subsomptions** sont de la forme : $C_1 \subseteq C_2$ avec C_1,$C_2 \in \mathbb{C}$. On dit alors que C_1 est un sous-concept de C_2 et que C_2 est un super-concept de C_1 ou $P_1 \subseteq P_2$ avec P_1,$P_2 \in \mathbb{P}$. On dit alors que P_1 est une sous-propriété de P_2 et que P_2 est une super-propriété de P_1. Par exemple, pour expliciter que le concept *Homme* est un sous-concept du concept *Humain* on note *Homme* $\subseteq$ *Humain* ou pour expliciter que la propriété *A-Fille* est une sous-propriété de la propriété *A-Enfant* on note *A-Fille* $\subseteq$ *A-Enfant*
2. les axiomes de **typage de propriétés**, i.e. la définition pour chaque propriété de son *domaine* et de son *image*. On note $D \triangleleft P \triangleright R$ l'axiome de *typage* de P ($P \in \mathbb{P}$) avec D le domaine de P tel que $D \subseteq \mathbb{C}$ et R l'image de P tel que $R \subseteq \mathbb{C} \cup \mathbb{T}$ avec $\mathbb{T}$ un ensemble de types simples (entiers, chaînes de caractères ...).
3. les **équivalences** sont de la forme : $C_1 \equiv C_2 \wedge ... \wedge C_n \wedge P_1 \wedge ... \wedge P_m$ avec C_1 ... $C_n \in \mathbb{C}$ et P_1 ... $P_m \in \mathbb{P}$. Par exemple, pour expliciter que le concept *Homme* est équivalent à la conjonction des concepts *Humain* et *Mâle* on note *Homme* $\equiv$ *Humain* $\wedge$ *Mâle* ou pour expliciter que le concept *Père* est équivalent à la conjonction du concept *Homme* et de l'ensemble des individus *Humain* possédant une valeur pour la propriété *A-Enfant*, on note *Père* $\equiv$ *Homme* $\wedge \exists$ *A-Enfant*.*Humain*.

DEFINITION 4: Un **individu** i_C ($i_C \in \mathbb{I}$) est une instance d'un concept C. On définit $\mathfrak{I}^{\mathbb{C}}$ une fonction qui associe à chaque concept C un ensemble d'individus telle que $\mathfrak{I}^{\mathbb{C}} : \mathbb{C} \rightarrow \mathbb{I}$. On écrit alors $i_C \in \mathfrak{I}^{\mathbb{C}}(C)$ et $\mathfrak{I}^{\mathbb{C}}(C) \subset \mathbb{I}$. Par exemple, si un concept *Homme* contient les individus *mathieu*, *nicolas*, *pierre* et *thomas* alors on note $\mathfrak{I}^{\mathbb{C}}$(*Homme*) = {*nicolas*, *mathieu*, *sami*, *thomas*}.

DEFINITION 5: Une **instance de propriété** i_P lie deux individus i_{C_x} et i_{C_y} ou un individu i_{C_x} et une valeur d'un type simple v. On définit $\mathfrak{I}^{\mathbb{P}}$ une fonction qui associe à chaque propriété P un ensemble de couples d'individus ou de couples individu/valeur telle que $\mathfrak{I}^{\mathbb{P}} : \mathbb{P} \rightarrow \mathbb{I} \times (\mathbb{I} \cup \mathbb{V})$ avec $\mathbb{V}$ un ensemble de valeurs d'un type simple. On écrit alors $(i_{C_x}, i_{C_y}) \in \mathfrak{I}^{\mathbb{P}}(P)$ ou $(i_{C_x}, v) \in \mathfrak{I}^{\mathbb{P}}(P)$ avec $\mathfrak{I}^{\mathbb{P}}(P) \subset \mathbb{I} \times (\mathbb{I} \cup \mathbb{V})$. Par exemple, on peut avoir $\mathfrak{I}^{\mathbb{P}}$(*Encadrant*)= {(*anne-Laure*, *sami*), (*véronique*, *mathieu*)}. Dans la suite, nous noterons $i_P(i_{C_x}, i_{C_y})$ pour $(i_{C_x}, i_{C_y}) \in \mathfrak{I}^{\mathbb{P}}(P)$. Par exemple, pour (*véronique*, *mathieu*) $\in \mathfrak{I}^{\mathbb{P}}$(*Encadrant*) on notera *Encadrant*(*véronique*, *mathieu*).

Dans la littérature portant sur les ontologies on trouve souvent les trois dénominations *relation*, *attribut* et *propriété*. Les deux premières dénominations sont des cas particuliers de la troisième. Une *relation* est une propriété qui lie deux individus, par exemple *Ami(jean,marie)* alors qu'un *attribut* est une propriété qui lie un individu à une valeur d'un type simple, par exemple *Age(marie,25)*.

On définit alors $\mathbb{P}$ comme une partition contenant les deux ensembles disjoints $\mathbb{P}_R$ et $\mathbb{P}_A$ définis comme suit :

- $\mathbb{P}_R$ l'ensemble des propriétés P de type *relation* telles que $D \triangleleft P \triangleright R$ avec $D \subseteq \mathbb{C}$ et $R \subseteq \mathbb{C}$ et $\mathfrak{I}^{\mathbb{P}}(P) \subseteq \mathbb{I} \times \mathbb{I}$
- $\mathbb{P}_A$ l'ensemble des propriétés P de type *attribut* telles que $D \triangleleft P \triangleright R$ avec $D \subseteq \mathbb{C}$ et $R \subseteq \mathbb{T}$ et $\mathfrak{I}^{\mathbb{P}}(P) \subseteq \mathbb{I} \times \mathbb{V}$

DEFINITION 6: Une **ontologie** est constituée d'ensembles de concepts, de propriétés et d'individus. Les concepts et les propriétés sont définis à l'aide d'axiomes terminologiques. Les individus sont associés aux concepts et propriétés à l'aide de fonctions d'association.

Une structure d'ontologie $\mathbb{O}$ peut alors être représentée par un tuple $< \mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}} >$ 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 d'individus et de valeurs.

Une autre vision des ontologies est celle issue du paradigme des logiques de description, qui y distingue une partie terminologique qui définit les éléments conceptuels qui composent l'ontologie et une partie assertionnelle qui déclare les individus du domaine et explicite leurs valeurs de propriétés. Dans cette vision une ontologie $\mathbb{O}$ est constituée d'un ensemble terminologique et d'un ensemble assertionnel, respectivement notés *T-Box* et *A-Box*.

$$\textit{On note } \mathbb{O} = \textit{T-Box} \sqcup \textit{A-Box}$$

La *T-Box* correspond aux ensembles de concepts $\mathbb{C}$, de propriétés $\mathbb{P}$ et d'axiomes terminologiques $\mathbb{A}$.

$$\textit{T-Box} = \mathbb{C} \sqcup \mathbb{P} \sqcup \mathbb{A}$$

La *A-Box* correspond aux ensembles d'individus et aux deux fonctions d'association $\mathfrak{I}^{\mathbb{C}}$ et $\mathfrak{I}^{\mathbb{P}}$.

$$\textit{A-Box} = \mathbb{I} \sqcup \mathfrak{I}^{\mathbb{C}} \sqcup \mathfrak{I}^{\mathbb{P}}$$

Dans la suite du manuscrit, lorsque l'on parle d'instances sans précision, nous faisons référence aux instances de concepts et de propriétés.

Tout au long du manuscrit, nous utiliserons les notations suivantes :

- les noms de concept et de propriété sont en *italique* et commencent par une majuscule ;

- les noms d'instances de concept sont en *italique* et commencent par une minuscule ;
- les concepts sont notés C_x et les instances de concept i_{C_x} ;
- les propriétés qui lient des concepts sont notées $P(C_x, C_y)$;
- les instances de propriété sont notées $i_P(i_{C_x}, i_{C_y})$ ou i_P (forme abrégée).

2.1.3 *OWL : le standard du Web sémantique*

Depuis 2004, le langage d'ontologie Web OWL est le format recommandé par le W3C pour représenter les ontologies. OWL a été créé dans le souci de répondre aux exigences principales attendues d'un langage d'ontologie [Antoniou *et al.*, 2003] qui sont :

- une syntaxe bien définie ;
- une sémantique bien définie ;
- un support de raisonnement efficace ;
- une puissance d'expression suffisante ;
- une simplicité d'expression.

Le développement de OWL a été soumis à une variété d'influences [Horrocks, 2005], incluant des influences de formalismes établis et de paradigmes de représentation des connaissances, des influences de langages d'ontologie existants, et des influences de langages du Web sémantique existants. Les logiques de description [Brachman et Levesque, 1984] ont eu une forte influence sur la conception du langage OWL, en particulier sur la formalisation de la sémantique, le choix de ses constructeurs, ainsi que l'intégration des types et valeurs de données. La structure de OWL a été influencée par le paradigme des frames [Minsky, 1974] et sa syntaxe a été conçue avec la nécessité de maintenir une compatibilité ascendante avec le langage Web *Resource Description Framework* (RDF) [Manola et Miller, 2004] et en particulier *RDF Schema* (RDFS) qui inclut plusieurs des caractéristiques de base d'un langage d'ontologie.

Tout comme RDFS, OWL permet de déclarer des concepts et des propriétés, les organiser hiérarchiquement et définir les domaines et images des propriétés. Les domaines des propriétés OWL sont des concepts OWL, et les images sont des concepts OWL ou des types simples pré-définis telles que des chaînes de caractères ou des entiers. OWL étend les capacités RDFS ; les concepts OWL peuvent être spécifiés comme des combinaisons (intersections, unions ou compléments) d'autres concepts, ou des énumérations d'individus. OWL permet d'affirmer qu'une propriété est transitive, symétrique, fonctionnelle, ou est l'inverse d'une autre propriété. Des déclarations d'équivalence et de disjonction peuvent être faites entre les concepts et les propriétés. L'égalité et l'inégalité entre les individus peut être déclarée.

De plus, OWL permet de définir des restrictions sur les propriétés portant sur leur domaine ou leur image. Il permet de définir des propriétés dont le type de valeur de tous les individus d'un même concept est restreint à un concept ou un type simple, ainsi que de définir les valeurs de cardinalités possibles ou nécessaires (au moins une valeur d'un certain concept ou type simple ; au moins certaines valeurs spécifiques ou au moins ou au plus, un certain nombre de valeurs distinctes)

L'exemple ci-dessous, adapté d'un exemple donné dans [Horrocks *et al.*, 2003] illustre les capacités de représentation de OWL.

OWL tout comme RDFS permet de :

- déclarer des concepts tels que *Pays*, *Personne*, *Étudiant* et *Français* ;
- déclarer que *Étudiant* est un sous-concept de *Personne* ;
- déclarer que *france* et *angleterre* sont des instances du concept *Pays* ;
- déclarer *Nationalité* comme une propriété qui relie les concepts *Personne* (son domaine) et *Pays* (son image) ;
- déclarer que *Age* est une propriété, qui a *Personne* comme domaine et *entier* comme image ;
- déclarer que *pierre* est une instance du concept *Français*, et que son *Age* a comme valeur *48*.

OWL permet en plus de :

- déclarer que *Pays* et *Personne* sont des concepts disjoints ;
- déclarer que *france* et *angleterre* sont des individus distincts ;
- déclarer *Citoyen* comme la propriété inverse de *Nationalité* ;
- déclarer que le concept *Apatride* est défini précisément comme les membres du concept *Personne* qui n'ont pas de valeurs pour la propriété *Nationalité* (cf. Figure 3) ;
- déclarer que le concept *MultiNational* est défini comme les membres du concept *Personne* qui ont au moins 2 valeurs pour la propriété *Nationalité* (cf. Figure 4) ;
- déclarer que le concept *Français* est défini comme les membres du concept *Personne* qui ont *france* comme valeur de propriété *Nationalité* (cf. Figure 5) ;
- déclarer que *Age* est une propriété fonctionnelle (qui ne peut posséder qu'une seule valeur).

OWL fait la distinction entre les deux types de propriété : relation et attribut.

- les relations sont appelées *object properties*.
- les attributs sont appelés *datatype properties*.

```
<EquivalentClasses><Class IRI="Apatride">
    <ObjectIntersectionOf>
        <Class IRI="Personne">
        <ObjectComplementOf>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="Nationalité">
                <Class IRI="Pays">
            </ObjectSomeValuesFrom>
        </ObjectComplementOf>
    </ObjectIntersectionOf>
</EquivalentClasses>
```

FIGURE 3: Définition du concept *Apatride*

```
<EquivalentClasses><Class IRI="MultiNational">
    <ObjectIntersectionOf>
        <Class IRI="Personne">
        <ObjectMinCardinality cardinality="2">
            <ObjectProperty IRI="Nationalité">
            <Class IRI="Pays">
        </ObjectMinCardinality>
    </ObjectIntersectionOf>
</EquivalentClasses>
```

FIGURE 4: Définition du concept *MultiNational*

```
<EquivalentClasses><Class IRI="Français">
    <ObjectIntersectionOf>
        <Class IRI="Personne">
        <ObjectHasValue>
            <ObjectProperty IRI="Nationalité">
            <NamedIndividual IRI="france">
        </ObjectHasValue>
    </ObjectIntersectionOf>
</EquivalentClasses>
```

FIGURE 5: Définition du concept *Français*

Le langage OWL se décline en trois versions : OWL-Lite, OWL-DL et OWL FULL. OWL-Lite est le plus simple et n'utilise qu'une partie des caractéristiques d'OWL. OWL-DL (DL désigne les logiques de description) a été conçu pour supporter les logiques de description et permettre un raisonnement décidable. OWL Full correspond au langage complet, qui relâche certaines contraintes de OWL-DL afin de rendre la représentation des connaissances plus flexible, cela au détriment de la décidabilité du raisonnement.

Notre approche étant fondée sur l'exploitation des mécanismes d'inférence fournis par le formalisme logique des ontologies, nous avons choisi OWL-DL comme langage de représentation. Ainsi, tout au long du manuscrit la mention de OWL fait référence à OWL-DL.

2.1.4 *Mécanismes d'inférence et de vérification*

La correspondance de OWL-DL avec les logiques de description (DL) et leurs propriétés permet l'exploitation d'algorithmes et de raisonneurs existants [Horrocks, 2005]. Plusieurs prototypes d'outils et d'applications pouvant faire usage des mécanismes de raisonnement des logiques de description ont précédé l'existence de OWL tels que FaCT [Horrocks, 1998], Racer [Haarslev et Möller, 2001] et Pellet [Sirin *et al.*, 2007].

Cette section a pour objectif de décrire les différents types de raisonnement que les logiques de description autorisent. On peut distinguer deux types de raisonnement : la déduction de nouvelles connaissances et la vérification de la cohérence des connaissances modélisées. Le raisonnement ontologique porte sur deux niveaux. Le niveau terminologique de l'ontologie (T-Box) et le niveau assertionnel (A-Box). Le raisonnement sur la T-Box permet d'inférer de nouveaux axiomes terminologiques et de vérifier la cohérence des axiomes terminologiques définis au sein de l'ensemble $\mathbb{A}$. Le raisonnement sur la A-Box permet d'une part d'inférer les classes sémantiques des individus de l'ontologie et de déduire des propriétés sur des individus et d'autre part de vérifier la cohérence des assertions faites sur les individus par rapport au cadre terminologique défini par $\mathbb{A}$.

Dans nos travaux, nous nous intéressons plus particulièrement au niveau assertionnel car il porte sur l'interprétation du monde. Cette interprétation sera définie par l'ensemble des instances qui seront mentionnées dans les spécifications d'exigences.

Le raisonnement sur le niveau terminologique s'appuie sur les quatre propriétés suivantes :

- Subsomption : un concept C_x est subsumé par un concept C_y si $\mathfrak{I}^{\mathbb{C}}(C_x) \subseteq \mathfrak{I}^{\mathbb{C}}(C_y)$

- Équivalence : deux concepts C_x et C_y sont équivalents si $\mathfrak{I}^{\mathbb{C}}(C_x) = \mathfrak{I}^{\mathbb{C}}(C_y)$.
- Disjonction : deux concepts C_x et C_y sont disjoints si $\mathfrak{I}^{\mathbb{C}}(C_x) \cap \mathfrak{I}^{\mathbb{C}}(C_y) = \emptyset$.
- Satisfaisabilité : un concept C est satisfaisable si $\mathfrak{I}^{\mathbb{C}}(C) \neq \emptyset$. C'est à dire qu'il existe au moins un individu du monde qui peut appartenir à l'ensemble décrit par ce concept.

Au niveau assertionnel, le raisonnement permet d'inférer les classes sémantiques et l'identité des individus ainsi que de vérifier leur cohérence. Dans le chapitre précédent (cf. section 1.8.4), nous avons défini les quatre types de propriétés (nécessaire, suffisant, nécessaire et suffisant et unicité) entrant dans le processus de classification et d'identification d'un individu. Nous entendons par *classification* d'un individu la détermination du concept auquel il appartient et par *identification* d'un individu sa distinction par rapport aux autres individus. Ci-dessous nous détaillons la formulation des axiomes terminologiques permettant la définition de telles propriétés.

2.1.4.1 *Classification des individus*

Le mécanisme de classification d'individus fait appel à deux types de connaissances : les axiomes qui définissent les conditions de classification et les assertions sur lesquelles s'appliquent ces conditions. La phase de classification consiste à associer les individus de l'ontologie aux concepts qui leur correspondent. Les axiomes en cause peuvent être définis de plusieurs façons sous OWL. Dans la suite, nous décrivons les mécanismes de raisonnement qui s'appliquent aux propriétés qui permettent la classification des individus de l'ontologie.

PROPRIÉTÉS NÉCESSAIRES L'application du raisonnement sur ce type de propriétés ne mène pas à la classification d'individus, cependant il permet la vérification de son résultat. Ce type de propriété est explicité par un axiome d'inclusion dans la description du concept. L'axiome correspondant est de la forme :

$$C_x \subseteq (C_i \cap ... \cap C_n) \cap ((\mathtt{card}_j P_j).C_j \cap ... \cap (\mathtt{card}_m P)_m.C_m)$$

Avec $C_i \cap ... \cap C_n$ l'intersection d'un ensemble de concepts et $\mathtt{card}_k$ une restriction de cardinalité sur une propriété P_k. L'axiome indique que l'ensemble des individus du concepts C_x doivent nécessairement appartenir à l'intersection des concepts $C_i \cap ... \cap C_n$ et posséder les propriétés $P_j, ..., P_m$ avec respectivement les restrictions de cardinalités $\mathtt{card}_j$, ..., $\mathtt{card}_m$ sur les classes sémantiques $C_j, ...C_n$ comme images. Par exemple, on peut définir le concept *Homme* comme l'ensemble des individus appartenant au concept *Humain* qui ne sont pas des individus du concept *Femme* (cf. 1)) et le concept *Père* comme

l'ensemble des individus appartenant au concept *Homme* possédant au moins un enfant (cf. (2)).

1. $\mathtt{Homme} \subseteq \mathtt{Humain} \cap \neg\mathtt{Femme}$
2. $\mathtt{Père} \subseteq \mathtt{Homme} \cap (\geqslant 1)$ *A-Enfant.Humain ou*

Dans l'axiome 1) il est aussi question de propriétés à satisfaire, car le concept *Homme* hérite des propriétés définies sur le concept *Humain* et ne devra pas satisfaire les propriétés du concept *Femme*.

PROPRIÉTÉS SUFFISANTES L'application du raisonnement sur ce type de propriétés permet d'inférer la classe sémantique d'un individu. OWL ne permet pas d'expliciter un axiome définissant l'ensemble des propriétés suffisantes dans la description du concept. Cependant, il fournit deux types d'axiomes permettant de définir des propriétés suffisantes à la classification d'un individu.

1. les propriétés *is-a* décrivent les relations hiérarchiques entre concepts. Elles permettent l'inférence suivante : si une instance i appartient au concept C_x, et C_x est une sous-classe de C_y, alors i est une instance de C_y.
$$i \in C_x \wedge C_x \subseteq C_y \Rightarrow i \in C_y$$
2. les domaine et image d'une propriété permettent de restreindre sa définition. De plus, lorsqu'ils sont définis, leur exploitation permet d'inférer la classe sémantique de certains individus de la manière suivante :
Soit P une propriété définie sur le domaine C_D et l'image C_R ($C_D \triangleleft P \triangleright C_R$) et i_1 et i_2 deux individus alors
$$P(i_1, i_2) \Rightarrow i_1 \in C_D \textit{ et } i_2 \in C_R$$

PROPRIÉTÉS NÉCESSAIRES ET SUFFISANTES Ces propriétés définissent les conditions nécessaires et suffisantes (CNS) pour appartenir à un concept. Ce type de propriété est explicité par un axiome d'équivalence dans la description du concept. L'axiome correspondant est de la forme
$$C_x \equiv (C_i \cap ... \cap C_n) \cap ((\mathtt{card}_j P_j).C_j \cap ... \cap (\mathtt{card}_m P)_m.C_m)$$

avec $C_i \cap ... \cap C_n$ l'intersection d'un ensemble de concepts et $\mathtt{card}_k$ une restriction de cardinalité sur une propriété P_k. L'axiome indique que chaque individu du concept C_x peut d'une part, être identifié

comme tel, s'il satisfait le membre droit et d'autre part, qu'il doit nécessairement satisfaire le membre droit.

Par exemple, on peut définir le concept *Homme* comme équivalent à l'ensemble des individus du concept *Humain* qui n'appartiennent pas au concept *Femme* (cf. (1)) ou le concept *Parent* comme équivalent à l'ensemble des individus du concept *Humain* qui possèdent une valeur de propriété *A-Enfant* définie sur l'image *Humain* (cf. (2)).

1. $\mathtt{Homme} \equiv \mathtt{Humain} \cap \neg \mathtt{Femme}$
2. $\mathtt{Parent} \equiv \mathtt{Humain} \cap (\exists)$ *A-Enfant.Humain*

2.1.4.2 *Identification des individus*

Les logiques de description sur lesquelles est fondé OWL ne supposent pas l'unicité des noms. C'est-à-dire que deux noms différents ne correspondent pas nécessairement à deux individus distincts. Identifier les individus similaires permet de fusionner leurs informations afin de limiter la redondance et faire ressortir les incohérences possibles. Sous OWL, les individus identiques sont représentés à l'aide de la propriété prédéfinie *SameAs*. Cette propriété permet d'exprimer que deux individus sont identiques. À l'inverse, la propriété *DifferentFrom* permet d'exprimer que deux individus sont différents.

À l'origine, la propriété *SameAs* a été conçue pour permettre d'unifier un même individu qui serait nommé de différentes manières au travers de multiples documents ou ressources. En effet, la richesse de la langue fait que dans différentes ressources, divers noms peuvent être utilisés pour faire référence à une même entité.

L'inférence de *SameAs* doit exploiter les propriétés qui, à l'instar des clés primaires définies en base de données ou de manière plus générale des contraintes d'unicités que l'on trouve dans les modèles de données, permettent d'identifier de manière unique les individus. Ci-dessous, nous décrivons deux mécanismes d'inférence pour la création d'une instance de propriété *SameAs* entre deux individus.

PROPRIÉTÉS FONCTIONNELLE ET FONCTIONNELLE INVERSE

OWL permet de spécifier les caractéristiques des propriétés à l'aide d'identifiants spéciaux qui sont utilisés pour fournir des informations sur les propriétés et leurs valeurs. Il offre alors des mécanismes puissants qui améliorent le raisonnement sur une propriété. Les propriétés fonctionnelle et inverse fonctionnelle sont de ces spécifications qui permettent d'augmenter les inférences possibles.

Une *propriété fonctionnelle* associe à chaque individu défini sur son domaine, une valeur d'image unique. Deux individus distincts peuvent alors être inférés comme étant identiques sur la base de leur valeur d'image de propriété commune. Par exemple, si la propriété *Num-Sécu* est conceptualisée comme fonctionnelle, deux individus ayant la même valeur d'image à travers elle, seront considérés comme identiques. Plus formellement, soit P une propriété fonctionnelle, i_j, i_k deux individus et i_i un individu ou une valeur simple, alors :

$$P(i_j, i_i) \wedge P(i_k, i_i) \rightarrow SameAs(i_j, i_k)$$

Une *propriété fonctionnelle inverse* associe à chaque individu de son image, une valeur de domaine unique. Deux individus distincts peuvent alors être inférés comme étant identiques sur la base de leur valeur de domaine de propriété commune. Par exemple, si la propriété *Père-Biologique* est conceptualisée comme inverse fonctionnelle, et si un même individu possède deux valeurs de domaine, ces valeurs seront considérées comme identiques. Plus formellement, soit P une propriété fonctionnelle inverse, i_j, i_k deux individus et i_i un individu ou une valeur simple, alors :

$$P(i_i, i_j) \wedge P(i_i, i_k) \rightarrow SameAs(i_j, i_k)$$

RÈGLES SWRL Parfois, plus d'une propriété peut être nécessaire pour identifier de manière unique un individu. Auquel cas, la contrainte d'unicité doit définir les valeurs de propriétés que deux individus doivent partager pour être inférés comme identiques. Cela peut s'apparenter en base de données aux clés primaires composées. Cependant, OWL ne permet par de définir des contraintes d'unicité portant sur la conjonction de plusieurs propriétés. Pour définir ce type de contrainte, il est possible de faire appel à des règles SWRL. SWRL est l'acronyme pour Semantic Web Rule Language. SWRL est fondé sur OWL, i.e. ses règles sont formées à partir des éléments d'OWL (concepts, propriétés, individus, littéraux).

Formellement, deux individus i_x et i_y sont identiques s'ils partagent les mêmes valeurs au travers de n propriétés P_1, .., P_n. Leur conjonction définit une contrainte d'unicité.

$$P_1(i_x, i_1) \wedge P_1(i_y, i_1) \wedge .. \wedge P_n(i_x, i_n) \wedge P_n(i_y, i_n) \rightarrow SameAs(i_x, i_y)$$

L'exécution des règles SWRL permet la création dynamique de propriétés. Nous y reviendrons lors de la discussion sur la représentation des règles.

2.1.5 *Conclusion*

Les ontologies permettent de définir les concepts et propriétés d'un domaine de manière formelle. Cette définition, lorsqu'elle est suffisamment 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. Dans ce travail de thèse, nous proposons d'explorer les possibilités de représentation et de vérification offertes par les ontologies OWL, pour la formalisation de spécifications en langage naturel.

2.2 MODÉLISATION D'UNE ONTOLOGIE

2.2.1 *Introduction*

Le développement d'une ontologie concerne sa conceptualisation ou son peuplement. Le premier consiste à développer la structure conceptuelle de l'ontologie ; il correspond à la définition de la *T-BOX*. Le second comprend la création de nouvelles instances de concepts ou de propriétés dans l'ontologie ; il correspond à la définition de la *A-Box*. Dans notre approche de la formalisation automatique de spécifications d'exigences, la conceptualisation de l'ontologie est une étape préalable à l'analyse des spécifications. Quant au peuplement, il correspond au résultat de l'analyse des spécifications.

Le développement d'une ontologie est une tâche difficile. Différents principes et méthodologies ont été proposés dans la littérature pour guider ce processus [Uschold et King, 1995; Gruber, 1995; Guarino, 1998]. L'ensemble pointe la nécessité d'identifier les concepts et propriétés clés du domaine, ainsi que les termes qui les désignent, puis d'en fournir une définition claire, précise et non ambiguë, le tout de façons formelle et consensuelle. Cette section a pour objectif de décrire la conceptualisation d'une ontologie dans ses grandes lignes, ainsi que de mettre en avant les caractéristiques requises pour faciliter le processus de peuplement.

Dans la littérature, il existe plusieurs propositions d'étapes de conceptualisation. Celle proposée par Buitelaar et Magnini [2005] décompose ce processus en six étapes illustrées en figure 6. Les six points décrits en partie gauche de la figure détaillent l'ordre d'acquisition des connaissances nécessaires à la conceptualisation de l'ontologie. La partie droite de la figure 6 illustre les couches successives correspondant aux connaissances extraites à chacune des étapes. La première couche correspond aux termes et synonymes et la dernière aux règles du domaine.

On peut distinguer deux types d'ontologies résultats d'une conceptualisation [Hernandez, 2005; Dasgupta *et al.*, 2013]. Les ontologies dites *légères* contiennent simplement une hiérarchie de concepts et une hiérarchie de propriétés. Les ontologies dites *lourdes*, sont enrichies par des axiomes utilisés pour fixer l'interprétation sémantique des concepts et des propriétés [Fürst et Trichet, 2006].

6. définition des règles.
5. définition des axiomes.
4. définition des propriétés non-taxonomiques entre les concepts.
3. hiérarchisation des concepts.
2. extraction des concepts.
1. extraction des termes et synonymes.

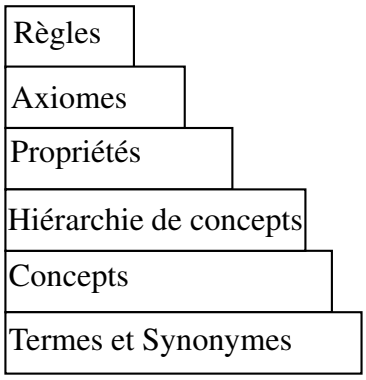


FIGURE 6: Étapes de conceptualisation d'une ontologie

La conceptualisation de l'ontologie concerne l'identification puis la définition de ses concepts, propriétés, axiomes. Ce processus demande une bonne connaissance du domaine. Ces connaissances sont détenues par des experts du domaine ou contenues dans des ressources, telles que les textes, taxonomies ou terminologies. On peut distinguer deux types de développement de l'ontologie : le développement entièrement manuel et le développement semi-supervisé.

2.2.2 *Développement manuel d'ontologie*

Aujourd'hui, la majorité des ontologies sont créées à la main, car les outils proposés ne permettent en général que de construire des ontologies *légères* [Hernandez, 2005], i.e. qui n'intègrent pas ou peu d'axiomes. Le développement automatique d'ontologies lourdes à partir de textes, a été bien moins abordé dans la littérature que celui des ontologies légères. La raison principale est que l'apprentissage d'ontologies lourdes dépend plus fortement de la compréhension des textes en langage naturel [Dasgupta *et al.*, 2013], ce qui demande un niveau d'analyse linguistique et sémantique plus poussé.

Le développement manuel d'ontologies peut se faire de deux façons : soit en rédigeant directement dans un langage d'ontologie, procédé qui requiert une bonne maîtrise de la syntaxe du langage et qui peut vite s'avérer fastidieux et inconfortable selon la taille de l'ontologie, soit le plus souvent en faisant appel à des outils d'édition d'ontologie. Ces éditeurs facilitent la création d'ontologie sans assis-

ter l'utilisateur dans sa tâche de conceptualisation. Depuis deux décennies un large nombre d'éditeurs d'ontologie ont été proposés tels que Protégé [3], KAON [4], SWOOP [5], NeOnToolKit [6].

2.2.3 *Développement (semi-)automatique d'ontologie*

Le développement (semi-)automatique d'ontologie prend le plus souvent comme point de départ des documents textuels. Le recours aux textes est légitimé d'une part par les travaux menés en linguistique dont l'hypothèse principale est que les textes sont porteurs de connaissances stabilisées et partagées par des communautés de pratiques [Mondary *et al.*, 2008] et d'autre part, par l'avancement des travaux de recherche dans le domaine du TAL, de l'extraction d'information et de l'apprentissage automatique qui permettent déjà l'extraction et la reconnaissance d'une grande quantité de connaissances présentes dans les textes.

La conceptualisation d'une ontologie à partir de textes ne peut s'effectuer de manière complètement non-supervisée, ce qui tient à la nature du langage naturel qui est plus un moyen de communication que de représentation, dont le sens des phrases dépend tout autant de la phrase que de son contexte d'énonciation [Russell *et al.*, 1996] et dont le style autorise de passer sous silence certaines connaissances acceptées et partagées. Les méthodes et outils proposent alors bien plus souvent une aide au développement pour réduire les coûts de temps et d'effort humain que demandent la conceptualisation d'une ontologie [Cimiano *et al.*, 2009] Une multitude d'outils ont été développés dans ce sens et intégrés à des outils d'édition d'ontologie, tel que Text-To-Onto [Maedche *et al.*, 2001] qui a été intégré à l'environnement d'ingénierie d'ontologie KAON, OntoLT [Buitelaar *et al.*, 2004] qui a été intégré à Protégé ainsi que Text2Onto [Cimiano et Völker, 2005] et Terminae [Szulman, 2011] qui ont été intégrés à NeOn Toolkit.

On peut distinguer deux catégories d'outils. Une première qui requiert une validation des éléments extraits, tels que Text2Onto [Cimiano et Völker, 2005] qui propose à l'expert les résultats finaux d'une extraction automatique portant sur des concepts, des relations de hiérarchie ou de méréologie et d'équivalence entre concepts ordonnés en fonction de mesures statistiques. L'expert parcourt alors les éléments qu'il est libre d'accepter ou de rejeter. OntoLT [Buitelaar *et al.*, 2004] est un plug-in Protégé qui intègre et permet de définir

3. http ://protege.stanford.edu/
4. http ://kaon.semanticWeb.org/
5. http ://code.google.com/p/swoop/
6. http ://neon-toolkit.org/wiki/Main

des règles d'association (XPATH) pour l'extraction automatique de concepts et d'attributs à partir de corpus XML annotés. L'activation de ces règles engendre la création de concepts et d'attributs candidats qui seront à valider par l'utilisateur via Protégé. Une seconde catégorie d'outils propose d'aider l'expert à extraire des éléments pertinents de manière interactive. OntoGen [Fortuna *et al.*, 2007] par exemple assiste l'utilisateur dans la construction d'une ontologie de thèmes où les instances de concepts sont des documents. L'utilisateur peut sélectionner un thème, et le système propose alors automatiquement ses sous-thèmes potentiels pondérés selon différents degrés de certitude. Le nombre de thèmes proposés est supervisé par l'utilisateur, qui sélectionne ensuite les sous-thèmes qu'il juge pertinents et le système les ajoute à l'ontologie comme sous-thèmes du thème sélectionné. Terminae [Szulman, 2011] est une plate-forme d'aide à la construction de ressources termino-ontologiques à partir de ressources textuelles, qui intègre des outils d'ingénierie linguistique et d'ingénierie des connaissances. L'outil permet à un utilisateur de lier des éléments textuels à des éléments ontologiques et vice-versa.

2.2.4 *Concept génériques et Concepts individuels*

Une des problématiques majeures dans la conceptualisation d'ontologie est la distinction entre individu et concept, car celle-ci ne va pas toujours de soi. Si elle dépend fortement de l'objectif de l'ontologie, elle peut dans certains cas être assez subjective. Par exemple, le terme *twingo* peut selon le choix de conception être une instance du concept *Voiture* (cf. figure 7 (1)), ou de son sous-concept *Voiture-Renault* (cf. figure 7 (2)) ou même un concept sous-concept de *Voiture* et *Voiture-Renault* qui lui-même pourrait posséder des individus tels que l'individu *twingo-AA-114-AA* (cf. figure 7 (3)).

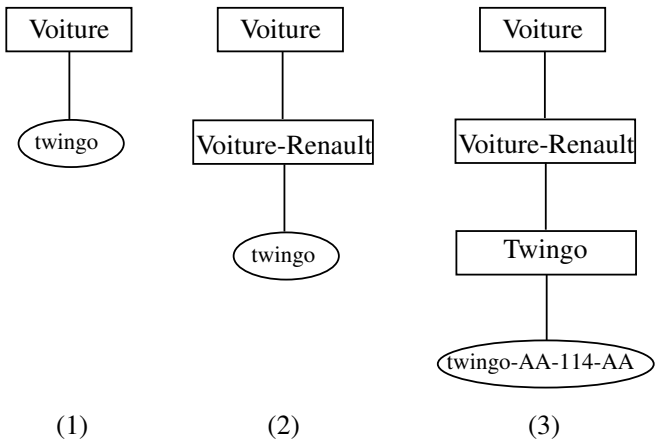


FIGURE 7: Représentations conceptuelles d'une twingo

Il est à remarquer que si le terme *twingo* peut faire référence à un individu, il ne représente pas pour autant une entité[7] réelle du monde mais fait plutôt référence à un type d'entité (les voitures qui sont du modèle Twingo), ce qui explique l'embarras du choix possible entre concept et individu. En revanche, si l'on considère l'individu *twingo-AA-114-AA* possédant le numéro d'immatriculation *AA-114-AA*, celui-ci ne peut en aucun cas correspondre à un concept car il ne peut représenter un type d'individu mais seulement une entité unique du monde (la voiture de type twingo immatriculée *AA-114-AA*). Ces deux exemples d'individus mènent à la distinction nécessaire de deux sortes d'individus au sein des ontologies, les individus qui représentent un type d'entités du monde et les individus qui représentent des entités du monde. Les individus sont définis par les concepts auxquels ils sont associés, aussi une distinction entre deux types d'individus découle nécessairement d'une distinction entre deux types de concepts.

Petasis *et al.* [2013] distinguent deux niveaux de concepts : les concepts de niveau intermédiaire (Mid-Level Concepts (MLC)) et les concepts de haut niveau (High Level Concepts HLC). Ils définissent les MLC comme des concepts qui peuvent être identifiés dans les corpus à partir de leur forme de surface et les HLC comme des concepts composés, formés à partir des MLC. Les HLC ne peuvent être directement identifiés à partir de corpus, car ils ne peuvent pas être associés à une forme de surface unique. Par exemple, le concept *Personne* est un HLC formé à partir de plusieurs MLC tel que *Prénoms*, *Age*, *Nationalité*, etc. Les instances des concepts MLC peuvent être identifiées à partir d'extracteurs d'information. En revanche, la reconnaissance des individus des HLC doit se faire à partir d'un raisonnement sur les instances des MLC.

Cette vision permet de distinguer deux types de concepts en fonction de leur modalité d'identification à partir d'un corpus. Cependant, cette vision ne permet pas d'expliciter la différence conceptuelle entre les concepts contenant des individus qui représentent des types d'entités (tel que *twingo*) et les concepts contenant des individus qui représentent des entités uniques (tel que *twingo-AA-114-AA*).

De même que leurs individus, les concepts au sein d'une ontologie se distinguent par leurs propriétés. Nous avons présenté en section 1.8.4 les quatre types de propriétés qui participent à la définition d'un concept : propriétés nécessaires, suffisantes, nécessaires et suffisantes et unicité. Ainsi, si l'on examine de plus près nos deux types d'in-

7. Il est d'importance à ce niveau de bien établir la distinction que nous faisons entre un individu et une entité. Tout au long de ce manuscrit, nous nommerons individu, toute instance de concept. Quant à la notion d'entité, nous y aurons recours pour faire à référence un objet réel du monde.

dividus, il s'avère que seuls les individus dénotant une entité réelle doivent posséder une propriété d'unicité de sorte à pouvoir être identifiés les uns par rapport aux autres de façon unique. Nous utilisons cette particularité pour expliciter la définition conceptuelle des deux types d'individus et nous inspirons de la terminologie des graphes conceptuels [Sowa, 1984] pour les nommer de la manière suivante : référent générique et référent individuel.

DEFINITION 7: Un **référent générique** désigne un type d'entités. Au sein de l'ontologie, il peut être classé à l'aide de ses valeurs de propriétés nécessaires et suffisantes.

DEFINITION 8: Un **référent individuel** désigne une entité du monde. Au sein de l'ontologie, il peut être classé à l'aide de ses valeurs de propriétés nécessaires et suffisantes et être identifié de manière unique à partir de ses valeurs de propriétés d'unicité.

Par extension, nous parlerons de *concept générique* et de *concept individuel* que nous définissons ci-dessous :

DEFINITION 9: Un **concept générique** définit un ensemble d'individus représentant des types d'entités du monde (des référents génériques). Il est défini par un ensemble de propriétés nécessaires et suffisantes.

DEFINITION 10: Un **concept individuel** définit un ensemble d'individus représentant des entités uniques du monde (des référents individuels). Il est défini par un ensemble de propriétés nécessaires et suffisantes ainsi que des propriétés d'unicité.

La distinction entre les concepts que nous proposons n'est pas équivalente à celle proposée par Petasis *et al.* [2013]. Nos concepts individuels sont nécessairement de haut niveau (HLC) (composés par d'autres concepts) au sens de *Petasis et al* néanmoins, les concepts génériques peuvent être indépendamment de haut niveau ou de niveau intermédiaire (MLC).

Lors de la conceptualisation, la définition des propriétés conceptuelles définissant un concept peut mettre en évidence le fait que celui-ci soit générique ou individuel, en fonction de ce concept qui peut ou doit posséder des propriétés d'unicité. La vision que nous proposons a pour objectif de lever le voile subjectif inhérent à la distinction entre concept et individu qui, dans le cas de référents individuels ne se posera pas et dans le cas de référents génériques dépendra de la spécialisation de l'ontologie voulue par son concepteur, i.e. la profondeur de l'ontologie.

2.2.5 *Profondeur de l'ontologie*

> Une ontologie est une spécification explicite du niveau de connaissance d'une conceptualisation [Van Heijst *et al.*, 1997].

Lors du développement conceptuel d'une ontologie, une autre question fondamentale concerne sa granularité, à savoir quelle profondeur hiérarchique est nécessaire à l'atteinte de l'objectif de l'ontologie. En d'autres termes il s'agit de 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, une taxonomie ou une terminologie listant un ensemble d'objets mais comme la structure au sein de laquelle catalogue, taxonomie 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. Construire une ontologie où chaque concept est clairement distingué par rapport aux autres demande une analyse très fine du domaine qui peut constituer un travail continuel.

La granularité choisie semble subjective. Elle est le plus souvent conditionnée par l'utilisation à laquelle l'ontologie est destinée. Dans notre contexte, ce qui importe est de pouvoir inférer de façon précise les classes sémantiques et l'identité des individus dont il est fait mention dans les spécifications d'exigences. Cela demande la définition pour chaque concept de l'ontologie, des propriétés nécessaires et suffisantes à l'inférence de la classe sémantique des individus et de l'identité des référents individuels. Une fois cette contrainte assurée, le concepteur est libre de spécialiser ou non les concepts présents. Cependant toute nouvelle définition de (sous-)concept doit s'accompagner de la définition de ses propriétés définissantes [Bouaud *et al.*, 1994].

Si nous reprenons notre exemple, la représentation de l'individu *twingo-AA-114-AA* pourrait correspondre à la définition des concepts *Voiture*, *Modèle-Voiture* et *Marque-Voiture*. Puis la définition des propriétés du concept *Voiture* se ferait comme suit : *Marque* (exemple *renault*), *Modèle* (exemple *twingo*) et *Immatriculation* (exemple *AA-114-AA*). Les deux premières propriétés permettraient la classification des individus de voiture quand la dernière permettrait leur identification.

2.2.6 *Conclusion*

Dans cette section, nous avons décrit dans ses grandes lignes le développement d'une ontologie. Celui-ci est une étape cruciale pour la modélisation d'un domaine et en particulier pour le processus de peuplement d'une ontologie. Nous avançons que le recours à une ontologie *lourde* où concepts et propriétés sont définis de manière précise et explicite peut simplifier le processus de peuplement et améliorer la qualité de son résultat. En outre, nous avons proposé une distinction entre deux niveaux de concepts qui s'étend à deux sortes d'individus : les individus représentant des objets du monde et les individus qui correspondent à des types de valeur permettant de caractériser les objets du monde. La prise en compte de cette distinction peut permettre de guider les choix entre concept et individu lors de la conceptualisation de l'ontologie. Dans la suite, nous montrons que cette distinction entre concepts génériques et concepts individuels joue un rôle important au sein du processus de peuplement. La section suivante offre une description détaillée des enjeux et des processus de peuplement d'ontologie. par rapport à l'état de l'art.

2.3 PEUPLEMENT D'ONTOLOGIE

L'objectif de cette section est de donner une vue globale des différentes approches constituant l'état de l'art du peuplement d'ontologie. Ces approches sont discutées sous plusieurs angles de sorte à faire émerger les problématiques liées au peuplement et de situer notre approche par rapport aux autres. Dans un premier temps, nous définissons le peuplement d'ontologie et décrivons ses étapes.

2.3.1 *Définition*

Conceptualiser une ontologie correspond à définir sa T-Box et permet de modéliser un monde. Le peuplement d'une ontologie suit généralement sa conceptualisation [Makki *et al.*, 2009] : cela correspond à y ajouter de nouvelles instances (remplir sa A-box) sans en changer la structure conceptuelle [Petasis *et al.*, 2011]. Il porte sur l'extraction et la classification des instances des concepts et des propriétés qui ont été définis dans l'ontologie [Ruiz-Martìnez *et al.*, 2011]. Le peuplement permet de donner une interprétation du monde représenté.

Plus formellement, le peuplement d'ontologie à partir de textes correspond à associer à un ensemble de termes $T = t_1, t_2, ..., t_n$ apparaissant dans un texte un ensemble de classes sémantiques définies par l'ontologie, i.e. les concepts $\mathbb{C}$, les propriétés $\mathbb{P}$ et les types simples $\mathbb{T}$.

Cela consiste à identifier pour chaque terme t_i l'instance i à laquelle il fait référence.

DEFINITION 11: *Un **terme** est une unité linguistique **désignant** un concept, un objet ou un processus. Le terme est l'unité de désignation d'éléments de l'univers perçu ou conçu. Il ne se confond que rarement avec le mot orthographique* [Gouadec, 1990]. *Dans une formulation traditionnelle, on dit qu'un **terme** descriptif **dénote** un ensemble d'entités et qu'il connote ou désigne une certaine propriété ou une certaine condition qu'une entité doit posséder ou remplir pour que le terme en question puisse lui être appliqué* [Chomsky et Dubois-Charlier, 1969] . En domaine de spécialité, les instances de concepts d'une ontologie sont dénotés dans les textes par des termes. Lors d'un processus de peuplement d'ontologie, on recherche chacun des termes mentionnés dans le texte qui puisse dénoter, ou faire référence à une instance de concept de l'ontologie. Peupler l'ontologie à partir de textes consiste alors à associer à chaque concept défini par l'ontologie, chacune de ses instances dénotée par une mention[8] dans le texte. L'approche de peuplement d'ontologie

In one traditional formulation a descriptive term is said to denote a set of entities and to connote or designate a certain property or condition that an entity must possess or fulfil if the term is to apply to it. [Chomsky, 1959].

8. Dans notre approche, nous considérons qu'une instance peut être dénotée de manière explicite ou implicite.

proposée au chapitre 3 considère que les termes peuvent aussi dénoter des instances de propriétés définies au sein de l'ontologie.

2.3.2 *Étapes du peuplement*

Petasis *et al.* [2011] proposent dans leur état de l'art sur le peuplement d'ontologie une méthodologie générale de peuplement, illustrée en figure 8. Cette figure indique que les entrées du processus de peuplement sont un corpus et une ontologie initiale. Ces entrées sont alors utilisées par un outil d'extraction d'instances qui donne en sortie l'ensemble des instances extraites. Les instances extraites servent alors au peuplement de l'ontologie initiale. Le résultat est une ontologie peuplée.

La méthodologie proposée possède deux inconvénients majeurs. Premièrement, elle ne fait pas apparaître l'aspect itératif du processus de peuplement et laisse supposer que le peuplement s'effectue nécessairement à partir d'une ontologie non préalablement instanciée. Deuxièmement, la place du raisonnement ontologique dans le processus de peuplement n'est pas explicitée. Pourtant le raisonnement ontologique peut permettre non seulement d'inférer les classes sémantiques et l'identité des nouveaux individus mais aussi de les valider. Nous proposons d'apporter quelques changements à la proposition de peuplement d'ontologie de Petasis *et al.* [2011] en y ajoutant l'aspect itératif du peuplement d'ontologie et la mise en avant des étapes d'inférence et de vérification nécessaires ainsi que la notion d'instance candidate. Notre proposition est illustrée en figure 9. Au fur et à mesure du peuplement, l'ontologie initiale évolue. Cette évolution doit être prise en compte à chaque nouvelle instanciation. Afin d'identifier les incohérences au plus tôt, à chacune de ses évolutions, l'ontologie doit être vérifiée. Si une nouvelle instance candidate introduit une incohérence, cette instance doit être rejetée (et soumise à l'utilisateur pour être révisée). Dans le cas contraire elle sera prise en compte lors d'une prochaine itération.

Dans la suite, nous discutons de l'ontologie initiale qui constitue le point de départ du processus de peuplement et des ressources termino-ontologiques, qui lui sont en général associées dans un processus de peuplement. Nous pointons la distinction qui existe entre les différents types d'instances recherchées. Puis, nous donnons une vue d'ensemble des méthodes proposées pour l'extraction d'instances candidates au peuplement. Enfin, nous comparons les approches de classification, d'identification et de validation des instances candidates issues d'une analyse des textes. Tout au long de notre discussion, nous positionnerons les notions de référents génériques et de référents individuels introduites au chapitre précédent (cf section 2.2.4)

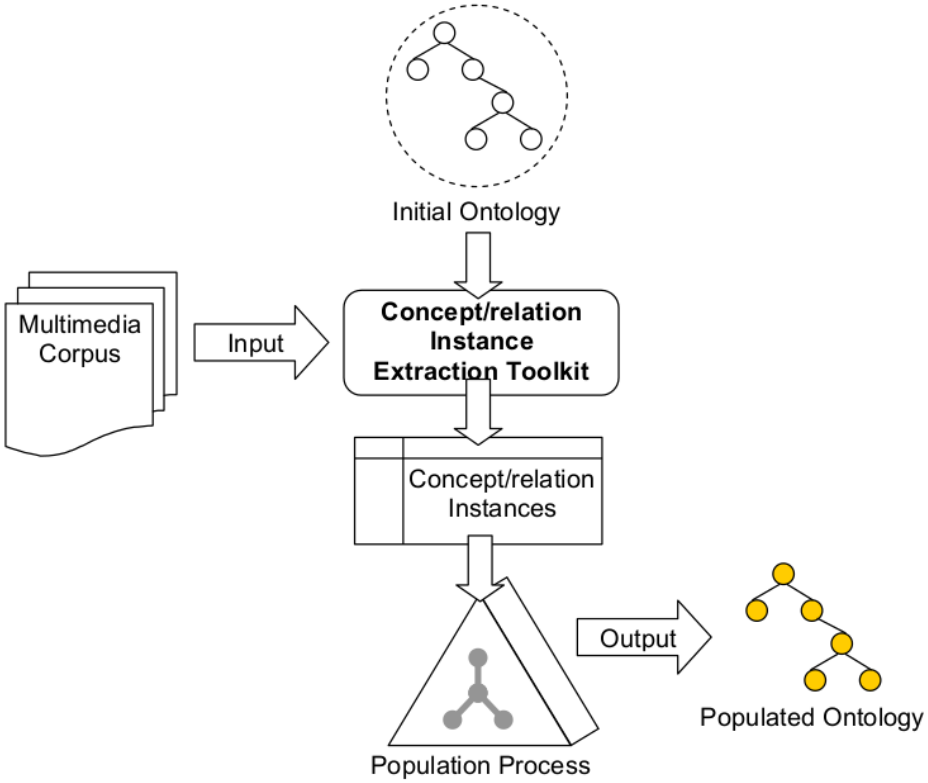


FIGURE 8: Processus de peuplement d'ontologie repris de Petasis *et al.* [2011]

par rapport à leurs usages dans l'état de l'art et nous montrons l'intérêt de guider le peuplement par l'identification d'instances de propriétés.

2.3.3 *Ontologie initiale*

Le peuplement d'une ontologie suppose l'existence d'une ontologie initiale. Cette ontologie représente le point de départ du processus de peuplement. Elle est donc le premier facteur dans l'orientation de l'extraction d'instances. Elle constitue l'ensemble des contraintes que les instances candidates au peuplement doivent satisfaire. Deux éléments distinguent les ontologies initiales : leur structure conceptuelle (légère ou lourde) et la possession d'instances de départ. Dans cette section, nous nous intéressons à l'impact de ces deux facteurs sur les méthodes de peuplement de l'ontologie.

STRUCTURE CONCEPTUELLE La structure conceptuelle d'une ontologie correspond à ses concepts, propriétés et axiomes terminologiques. Concepts et propriétés correspondent aux classes sémantiques à peupler ou instancier. Leur existence conditionne le processus de peuplement. Les axiomes terminologiques quant à eux définissent la

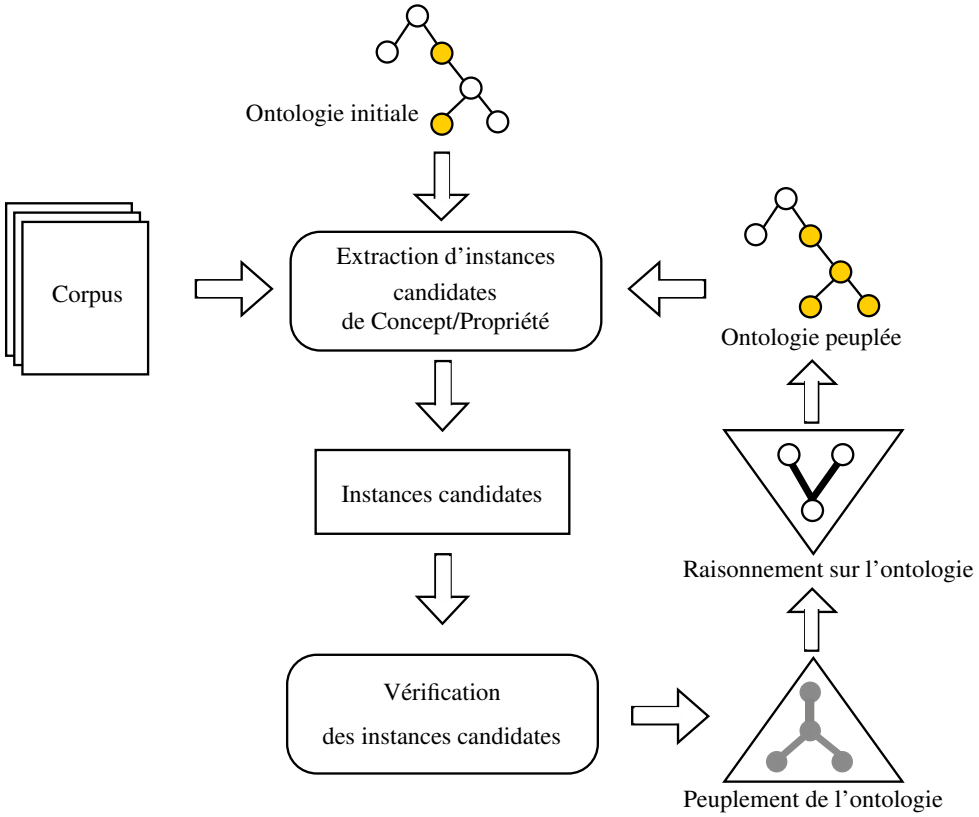


FIGURE 9: Peuplement d'ontologie

sémantique liée aux concepts et propriétés. Plus l'engagement sémantique choisi dans la formalisation est élevé, plus les problèmes liés à l'ambiguïté diminuent [Hernandez, 2005], ce qui permet de faciliter la reconnaissance des instances de concepts et de propriétés à partir des textes. De plus, la définition précise et formelle des concepts et propriétés de l'ontologie autorise l'application d'un raisonnement automatique. Ce raisonnement permet la classification et l'identification des individus spécifiés dans les textes, ainsi que la validation de la cohérence de l'ontologie résultante.

Peu de méthodes font état de l'utilisation d'ontologies lourdes. Certaines approches telles que celles de Ruiz-Martìnez *et al.* [2011] proposent une validation des instances par un raisonnement sur la cohérence de l'ontologie ; néanmoins les axiomes terminologiques n'y sont pas décrits. Petasis *et al.* [2013] propose une approche qui substitue une partie du procédé d'extraction d'instances par un raisonnement, guidé par un ensemble de règles terminologiques acquises automatiquement. L'objectif de ces règles est de déduire des instances de concepts de haut niveau (HLC) à partir d'instances de propriétés portant sur des instances de concepts de niveau intermédiaire (MLC).

La majorité des approches partent d'ontologies légères. La classification et l'identification des instances doit alors se faire hors de l'on-

tologie, de même que leur validation. Dans ce cas, des outils de classification et d'identification doivent être développés pour se substituer au raisonnement ontologique.

INSTANCES DE DÉPART Dans la plupart des cas, les ontologies sont partiellement peuplées pendant la phase de développement [Giuliano et Gliozzo, 2008]. Une ontologie à peupler contient un ensemble d'instances (possiblement vide au départ). Cet ensemble évolue au fur et à mesure du peuplement. Les instances contenues dans l'ontologie durant le peuplement d'ontologie sont soient issues de l'analyse des textes soit d'une instanciation préalable au processus de peuplement. Intuitivement, l'utilisation des instances présentes dans l'ontologie initiale ne peut être que pertinent. Elles représentent des connaissances qu'il est, si ce n'est nécessaire, du moins utile de prendre en compte. Premièrement, les instances initiales sont potentiellement liées aux nouvelles instances que l'on souhaite extraire des textes. Deuxièmement, les nouvelles instances doivent être consistantes avec celles déjà existantes. Enfin, le raisonnement sur les instances connues peut permettre d'inférer la classe sémantique ou les propriétés des nouvelles instances identifiées.

Certaines approches présupposent l'existence d'instances de concept [Thongkrau et Lalitrojwong, 2012], ou d'instances de concept et de propriétés[9] [Boer *et al.*, 2007] et utilisent ces instances dans un processus de reconnaissance de nouvelles instances dans les textes. Par exemple, Thongkrau et Lalitrojwong [2012] prennent en entrée une ontologie contenant des instances de chaque concept qu'ils nomment instances d'entraînement. Ces instances servent d'ensemble d'amorçage pour l'extraction de nouvelles instances de concept. Boer *et al.* [2007] proposent une approche qui suppose la connaissance de toutes les instances de concepts de l'ontologie de départ ainsi que quelques exemples de relations entre elles. Ces instances sont utilisées pour l'identification de nouvelles instances de relations à partir de documents textuels.

2.3.4 *Ressources termino-ontologiques*

L'un des enjeux majeurs du peuplement automatique d'ontologie à partir de textes est l'identification de mentions d'instances. Cela pose le problème de reconnaître dans les textes la mention d'une instance sous différentes formulations. Dans les textes de spécialité, ces mentions correspondent typiquement aux termes dénotant les instances.

9. Il est à noter que l'existence d'instance de propriétés dépend nécessairement de l'existence d'instances de concept

Être à même de reconnaître les différents termes qui dénotent une instance, suppose de disposer de ressources encodant des connaissances lexicales et linguistiques. Lors de la conceptualisation d'une ontologie formelle, les informations lexicales et linguistiques prises en considération ne sont pas toutes préservées dans la représentation finale de l'ontologie [Badra *et al.*, 2011] car les ontologies formelles ont pour vocation de représenter des connaissances d'ordre conceptuel et non lexical ou linguistique. D'autres formalismes sont en charges de représenter les connaissances lexicales et linguistiques, tels que les terminologies et les thésaurus.

Être en mesure de lier les termes spécifiés dans le texte aux instances conceptualisées dans l'ontologie demande de capturer la relation entre les constructions du langage naturel et les structures ontologiques, il est donc crucial d'associer connaissances linguistiques et ontologiques. En ce sens, divers travaux ont souligné l'intérêt de maintenir un lien entre connaissances terminologiques et ontologiques [Szulman *et al.*, 2010; Cimiano *et al.*, 2011]. Plusieurs chercheurs se sont intéressés à l'association d'une partie terminologique aux ontologies [Roche, 2008; Cimiano *et al.*, 2011; McCrae *et al.*, 2011; Ghersedine *et al.*, 2012] afin d'établir un lien entre les termes d'un texte et le vocabulaire conceptuel d'une ontologie. Roche [2008] le premier, introduit la notion de ressource termino-ontologique (RTO), distinguant les dimensions linguistiques (la terminologie) et conceptuelles (l'ontologie) ainsi que les relations entre elles. Une termino-ontologie est un thésaurus [10] dont les termes sont sémantiquement liés aux classes sémantiques définies dans une ontologie formelle [Roche *et al.*, 2009].

Parmi les modèles de représentation d'ontologies linguistiques les plus connus, on peut citer *SKOS* [11], *Lemon* [12], WordNet [13]. Ils sont fondés sur RDF et permettent tous deux d'associer des termes à leurs dénotations conceptuelles. Le Simple Knowledge Organization System (SKOS) permet de définir un ordre de préférence entre les termes référant à un même concept ou une même propriété à l'aide des annotations *préférence*, *alternative* et *cachée*. Lemon quant à lui permet la définition d'une plus large variété d'information linguistique. WordNet est l'un des lexiques les plus utilisés. Cependant, contrairement à SKOS et Lemon, il est structurellement indépendant du domaine, ce qui le rend peu adapté pour un rôle de termino-ontologie.

10. Les thésaurus augmentent les terminologies par un ensemble de relations sémantiques (relation de préférence par exemple).
11. http ://www.w3.org/TR/skos-reference/
12. http ://lemon-model.net/
13. http ://wordnet.princeton.edu/

2.3.5 *Type d'instances recherchées*

Selon l'objectif du peuplement d'ontologie, deux types d'instances peuvent être recherchés, les instances de concept (individus) et les instances de propriétés. Les individus se divisent eux-mêmes en deux catégories : les référents génériques et les référents individuels (cf. section 2.2.4). Si la distinction est claire entre la recherche d'individus et celle d'instances de propriétés, celle existant entre référents génériques et référents individuels n'est pas établie dans la littérature. La distinction que nous entendons ici : est ce que les individus extraits doivent être identifiés de façon précise ou seulement classés au sein de l'ontologie ? Cela correspond typiquement à faire la différence entre la recherche de noms-labels ou noms de types d'entités (référents génériques) ou la recherche d'entités du monde (référents individuels). Dans la littérature, une grande partie des approches s'intéresse au peuplement de référents génériques. Par exemple, le système Ontosophie [Celjuska et Vargas-vera, 2004] a pour objectif d'identifier dans les textes un ensemble de propriétés pouvant dénoter un concept. Pour chaque concept, un ensemble de règles d'extraction est appliqué afin d'identifier la présence de ses propriétés et de leur valeur. Une instance est alors construite à partir des propriétés extraites. Les propriétés qu'elle possède servent à la classer dans l'ontologie mais ne sont pas mises à profit pour la distinguer des autres instances de concept. Thongkrau et Lalitrojwong [2012] utilisent le même principe de classification fondé sur l'exploitation de valeurs de propriétés (majoritairement du type *is-a*) avec une approche plus statistique qui fait appel à une *analyse sémantique latente* [Deerwester *et al.*, 1990]. Les individus extraits ne correspondent pas à des entités du monde mais plutôt à des termes du domaine. Le même constat est à faire pour l'outil Onto-Text [Anantharangachar *et al.*, 2013] qui identifie à partir de textes des triplets représentant des propriétés d'instances de concept qui n'ont pas vocation à permettre l'identification des individus sur lesquels elles portent.

De manière générale, les approches qui portent sur le peuplement de référents individuels sont celles qui s'intéressent à la reconnaissance et la classification d'entités nommées (RCEN) [Cimiano et Völker, 2005; Magnini *et al.*, 2006; Giuliano et Gliozzo, 2008; Maynard *et al.*, 2009; Ruiz-Martìnez *et al.*, 2011]. En effet, les entités nommées ont la particularité de dénoter des entités du monde [Omrane *et al.*, 2012]. La reconnaissance et la classification d'entités nommées est une sous-tâche de l'extraction d'information [Ruiz-Martìnez *et al.*, 2011] qui vise à repérer et à classer les termes d'un texte en catégories prédéfinies telles que des noms de personnes, des organisations, des lieux ou des quantités. L'utilisation d'outils d'extraction d'entités nommées

pour le peuplement d'ontologie repose sur l'hypothèse que la mention d'une entité nommée d'une certaine catégorie dans un texte révèle l'existence de cette entité et permet de créer une instance du concept correspondant à cette catégorie dans l'ontologie [Magnini *et al.*, 2006]. Le peuplement d'ontologie est similaire à la reconnaissance et la classification d'entités nommées. La différence entre les deux est que le peuplement d'ontologie s'intéresse à des concepts à un niveau plus spécifique [Thongkrau et Lalitrojwong, 2012]. Selon Tanev et Magnini [2008], une différence majeure tient au fait que dans les RCEN, chaque occurrence d'un terme reconnu comme une entité nommée, i.e. un objet du monde, doit être classée séparément, tandis que pour le peuplement d'ontologie, c'est le terme en tant qu'unité linguistique et non ses occurrences qui doit être classé indépendamment du contexte dans lequel il apparaît. Cette vision du peuplement d'ontologie ne se limite qu'aux termes dénotant des individus de type référents génériques et occulte le fait que certains termes peuvent référer à des entités, aussi distinctement que peuvent le permettre des entités nommés [Mondary *et al.*, 2008]. Dans le cas de termes dénotant des référents individuels, le principe de reconnaissance est similaire à celui d'EN car ils représentent comme elles des entités du monde. Dans les textes, les EN sont un cas particulier de référents individuels.

Il ressort clairement que les deux types d'individus que nous distinguons, à savoir référent générique et référent individuel, ne se côtoient pas dans les approches de peuplement d'ontologie proposées dans la littérature. Or, dans la pratique, elles peuvent et parfois même doivent s'intégrer naturellement dans une même ontologie. C'est le cas dans l'approche de peuplement d'ontologie que nous proposons.

Un autre point important correspond à la précédence qu'il peut y avoir entre la reconnaissance d'individus et celle de leurs propriétés durant le processus de peuplement d'ontologie. Il apparaît que la classification des individus fait souvent appel à l'exploitation des propriétés et des valeurs de propriétés qu'ils possèdent. De plus, les entités (i.e. référents individuels) ne peuvent être identifiées directement à partir de mentions explicites dans les textes, seules leurs propriétés et valeurs de propriétés peuvent l'être [Castano *et al.*, 2008; Petasis *et al.*, 2011]. Il semble alors pertinent, dans une approche qui ambitionne d'extraire et d'identifier l'ensemble des types d'instances, de guider cette identification par celle des instances de propriétés.

2.3.6 *Méthodes d'extraction*

Le peuplement d'ontologie est une discipline assez récente. Néanmoins, l'avènement du Web sémantique a attisé l'intérêt porté à cette discipline. Il en résulte plusieurs propositions d'approches.

La majorité des approches de peuplement automatique d'ontologie à partir de textes sont fondées sur l'utilisation de méthodes d'extraction d'information (EI) [Cimiano et Völker, 2005; Yildiz et Miksch, 2007; Tanev et Magnini, 2008; Maynard *et al.*, 2008; Ruiz-Martìnez *et al.*, 2011; Faria *et al.*, 2013], d'apprentissage automatique (AA) [Cimiano et Völker, 2005; Tanev et Magnini, 2008; Carlson *et al.*, 2010] ou de techniques de traitement automatique du langage (TAL) [Cimiano et Völker, 2005; Maynard *et al.*, 2008; Tanev et Magnini, 2008; Ruiz-Martìnez *et al.*, 2011; Faria *et al.*, 2013]. Ces approches font appel à diverses méthodes, telles que l'extraction d'informations fondées sur les ontologies, l'apprentissage automatique par amorçage, la conception de grammaires locales ou encore la modélisation de contextes linguistiques. Dans cette section, nous discutons un certain nombre de travaux de peuplement d'ontologie au fil de ces méthodes.

2.3.6.1 *Extraction d'information fondée sur les Ontologies (EIBO)*

Le peuplement d'ontologie fait souvent appel à des techniques d'extraction d'information qui mettent à profit les connaissances de l'ontologie [Maynard *et al.*, 2008]. Il s'agit alors de guider la reconnaissance dans le texte des entités nommées ou des termes spécifiques au domaine par les connaissances ontologiques. Cette section donne un aperçu de ce type d'approche.

L'extraction d'information (EI) consiste à rechercher de façon automatique dans des textes, des informations d'un certain type. Les extracteurs utilisés doivent alors être guidés par des connaissances particulières aux types d'informations recherchées. Dans le cas du peuplement d'ontologie, les informations à extraire sont dépendantes des connaissances modélisées dans l'ontologie, car elles correspondent aux instances des classes sémantiques définies par celle-ci. De fait, guider l'extraction d'information par les connaissances modélisées par l'ontologie paraît naturel. L'extraction d'instances pour le peuplement d'ontologie est donc le plus souvent guidée par l'ontologie. Ce type d'approches et de systèmes est nommé système d'*extraction d'information fondée sur les ontologies* (EIBO) [Mokhtari, 2010]. Bien qu'assez récentes, ces approches ont déjà donné lieu à de nombreuses publications [Wimalasuriya et Dou, 2010]. Wimalasuriya et Dou [2010] proposent une architecture générale des systèmes EIBO et les classifient en fonction des méthodes d'extraction d'information qu'ils emploient, la manière dont ils acquièrent ou actualisent l'ontologie, les éléments de l'ontologie qu'ils cherchent à extraire ainsi que le type de ressources textuelles qu'ils traitent. Les EIBOs permettent une annotation des textes à partir du vocabulaire conceptuel de l'ontologie. Cela permet une comparaison directe des résultats extraits avec le vo-

cabulaire recherché. Les termes extraits sont alors plus à même de constituer des instances candidates pertinentes. Embley [2004] propose une des premières approches à considérer l'ontologie comme une partie intégrante d'un système d'extraction d'information. Il l'applique à l'extraction de concepts et relations à partir de documents Web afin de les associer à des ensembles de concepts et de relations décrits dans l'ontologie. Divers travaux ont proposé l'utilisation des EIBO pour guider le peuplement d'ontologie [Boer *et al.*, 2007; Yildiz et Miksch, 2007]. Les connaissances ontologiques sollicitées peuvent être de différents ordres. Boer *et al.* [2007] mettent à profit la connaissance des domaine et image des relations définis dans l'ontologie pour identifier la présence de la mention d'une instance de propriété. Ils utilisent les noms des instances de concept du domaine de cette propriété pour sélectionner un ensemble de documents à partir du Web, puis les noms des instances d'image de la propriété sont recherchés dans ces documents. Le système ontoX [Yildiz et Miksch, 2007] utilise une EIBO pour guider l'extraction d'attributs et de valeurs de types simple (entier, réel ...).

L'identification de mentions de règles de comportement, qui constitue notre objectif principal, n'a à notre connaissance, pas été abordé par l'utilisation d'EIBO.

L'identification des règles de comportement dans les textes n'est pas triviale car elle demande non seulement d'identifier les constituants (prédicats) de la règle mais aussi d'être en mesure de distinguer ses deux parties : antécédent et conséquent. Or, en pratique certains prédicats peuvent aussi bien apparaître dans l'antécédent que dans le conséquent. Par exemple, la valeur de l'état d'un appareil décrit par la propriété *État* peut correspondre à une condition à l'application d'une règle ou à la conséquence de son application. En plus de la reconnaissance des mentions de prédicats dans les textes, une autre difficulté en ce qui concerne la reconnaissance de règles de fonctionnement est la distinction de l'antécédent et du conséquent. Dans certains cas, des termes introducteurs peuvent être une bonne indication [Gordon et Harel, 2009; Guissé, 2013]. Cependant, ces termes ne sont pas toujours présents. Certaines approches fondent la reconnaissance de règles de comportement sur l'utilisation de modèles de représentation. Gordon et Harel [2009] définissent une grammaire fondée sur la structure de graphes LSC [14]. Cette grammaire décrit un cadre de connaissances communes permettant d'identifier la mention d'exigences dans les textes. Lorsqu'une phrase analysée peut être associée à plus d'une structure de base, l'utilisateur est sollicité pour

14. Un diagramme de séquences en direct (LSC) capture une partie de l'interaction entre les objets du système ou entre le système et son environnement. Il distingue les comportements possibles, nécessaires, ou interdits

trancher sur l'ambiguïté. De manière similaire, Kof [2009] fait usage de graphes MSC [15] comme référence pour valider des informations extraites de spécifications portant sur le comportement d'un système. Ainsi, le recours à des modèles de représentation peut permettre de guider l'identification de règles de comportement.

Un bon compromis dans l'identification automatique de règles peut alors tenir à la définition manuelle à partir de connaissances du domaine de patrons de règles (générales) du domaine, de sorte à définir un cadre suffisamment riche pour repérer des règles plus spécifiques aux exigences utilisateur.

2.3.6.2 *Apprentissage automatique pour le peuplement*

Dans le cadre du peuplement d'ontologie, l'apprentissage automatique consiste en la production empirique de règles d'extraction et de classification d'instances à partir des textes. On distingue trois types d'apprentissage automatique : supervisé, non supervisé et semi-supervisé. L'apprentissage supervisé consiste à produire automatiquement des règles à partir d'un ensemble de données d'apprentissage contenant des exemples validés par un expert. L'apprentissage non supervisé aussi nommé *clustering* consiste à diviser un ensemble de données hétérogènes, en sous-groupes de sorte que les données évaluées comme les plus similaires soient associées au sein d'un même groupe et qu'au contraire, les données évaluées comme différentes soient regroupées dans des groupes distincts. Contrairement à l'apprentissage supervisé, les données en entrée ne sont pas étiquetées et les catégories en sortie ne sont a priori pas connues. L'apprentissage semi-supervisé prend en entrée un ensemble de données étiquetées et non-étiquetées, généralement une petite quantité de données étiquetées avec une grande quantité de données non étiquetées. L'intérêt de cette technique est qu'elle minimise l'effort d'annotation qui peut s'avérer fastidieux dans le cas de grands jeux de données. De plus, si le modèle mixte est correct, l'utilisation de données non-étiquetées, en combinaison avec des données étiquetées, permet d'améliorer la qualité de l'apprentissage [Zhu, 2006; Chapelle *et al.*, 2010]. L'amorçage est une forme d'apprentissage semi-supervisé qui est conçu pour utiliser encore moins d'exemples d'apprentissage ; on dit souvent qu'il est faiblement supervisé. L'amorçage débute avec quelques exemples d'apprentissage, qu'il étend au fur et à mesure de ses itérations. Cette méthode est souvent utilisée dans le cadre du peuplement d'ontologie où l'on possède souvent au préalable des exemples d'instances de concepts [McDowell et Cafarella, 2008; Thongkrau et Lalitrojwong,

15. Un diagramme de séquences de messages (MSC) est un support visuel pour la description des scénarios qui capture l'interopérabilité des processus ou des objets.

2012] ou de propriétés [Boer *et al.*, 2007; Sun *et al.*, 2013]. En effet le pouvoir de représentation de l'ontologie peut amorcer la tâche d'apprentissage par l'exploitation des connaissances représentées [Valarakos *et al.*, 2004].

Afin de peupler une ontologie à partir de textes en langage naturel, le système *Ontosophie* [Celjuska et Vargas-vera, 2004] utilise une méthode d'apprentissage supervisée qui a comme point de départ un corpus annoté, dans lequel chaque EN est associée à une classe sémantique de l'ontologie. L'analyseur syntaxique *Marmot* [16] qui permet une analyse en parties du discours est ensuite appliqué au corpus. L'outil *Crystal* [16] [Soderland *et al.*, 1995] est ensuite utilisé pour générer des règles d'extraction à partir du corpus annoté. Les méthodes d'apprentissage automatique supervisé nécessitent un grand nombre d'annotations pour être capables d'apprendre des patrons d'extraction, ce qui limite leur utilisation dans le cas d'un grand nombre de concepts et de propriétés à considérer [Cimiano et Völker, 2005], car l'annotation demande alors un travail important. Ces approches ne sont alors pas aisément généralisables car en fonction de l'ontologie à peupler l'ensemble des classes sémantiques changent et si l'on considère des centaines de classes, une approche supervisée basée sur l'annotation d'un grand nombre d'exemples ne semble pas faisable. Cimiano et Völker [2005] proposent une approche non supervisée pour aborder le problème de la classification d'entités nommées au sein de grands ensembles de classes représentées par une ontologie. L'approche proposée exploite des caractéristiques lexico-syntaxiques ; considérées comme les plus efficaces, et est fondée sur la similarité contextuelle entre les concepts de l'ontologie et le terme à classer. Cette approche présente deux inconvénients majeurs : premièrement, le contexte distributionnel d'un terme peut être très différent de celui d'un nom de concept [Tanev et Magnini, 2008] ; de plus, l'utilisation du seul nom de concept limite la quantité de données d'apprentissage au nombre de concepts définis dans l'ontologie. Tanev et Magnini [2008] proposent d'améliorer cette dernière approche en comparant les termes non plus avec les noms de concepts mais avec des ensembles de termes dénotant chaque concept. Ils proposent un algorithme non supervisé basé sur une mesure de vecteur de similarité. L'algorithme est appliqué à un corpus syntaxiquement analysé contenant chaque entité d'entraînement au moins deux fois. Pour chaque occurrence dans le corpus, des traits syntaxiques sont relevés et utilisés pour construire un vecteur de caractéristiques. Ce nouveau vecteur est alors comparé aux vecteurs existants. La nouvelle instance est associée au concept possédant le vecteur de caractéristiques le plus proche. Giuliano et Gliozzo

16. Développé à l'université du Massachusetts, MA, USA

[2007] font appel à de l'inférence lexicale. Un terme est affecté à un concept si sa mention dans le texte peut être remplacée par la lexicalisation du concept. Pour Giuliano et Gliozzo [2008], la méthodologie diffère quelque peu : la substitution des termes dans les textes est faite non pas à partir des noms de concepts mais du nom de la désignation de leurs instances.

2.3.6.3 *Conception de grammaires locales*

Dans certains domaines spécifiques les documents rédigés peuvent parfois suivre un même style d'écriture. Les approches d'analyse des textes peuvent tirer partie de cette spécificité pour définir des grammaires fondées sur des structures de phrases prédictibles dans un domaine particulier. L'objectif de ces grammaires est de permettre la correspondance entre des éléments du texte et les éléments des patrons définis par des grammaires. La définition d'une grammaire permet d'éviter le recours à un formalisme sémantique intermédiaire [Fantechi *et al.*, 1994]. Xiao *et al.* [2012] proposent une approche pour l'extraction automatique de politiques de contrôle d'accès (PCA) à partir de textes en LN. Une inspection manuelle de 217 phrases portant sur des politiques de contrôle d'accès (PCA) a fait apparaître que 85% de ces phrases suivent le style [subject][can/cannot/is allowed to] [action][resource].

Dans le but de formaliser des spécifications d'exigences LN exprimant le comportement et les propriétés d'un système réactif vers le langage de logique temporelle (ACTL), Fantechi *et al.* [1994] proposent *NL2ACTL*, un outil fondé sur l'association d'une grammaire et d'un dictionnaire, qui permet une traduction automatique entre LN et ACTL. Dans l'optique de cette formalisation, une grammaire permettant la correspondance entre ces deux langages est construite à partir d'un corpus. Cette grammaire a comme objectif d'identifier des régularités dans les structures de phrases du corpus, tel que « *it is possible <action_phrase>* » ou « *if <action_phrase> then <sentence>* ». Les règles de grammaire sont appliquées dans le processus de reconnaissance de segments de phrases. Durant ce processus, l'intervention de l'utilisateur (celui qui écrit les exigences) est sollicitée en cas de sous-spécification ou d'ambiguïté.

L'outil NLForSpec proposé par Leitao *et al.* [2007] a pour objectif de permettre une traduction de cas de tests spécifiés en LN en une représentation formelle en CSP [17] (Communicating Sequential Processes). La méthode est fondée sur l'exploitation des catégories grammaticales des mots pour déterminer leur rôle. NLForSpec est composé de trois

17. CSP, est un langage de description de modèles d'interaction. Il correspond à une algèbre de processus permettant de modéliser l'interaction de systèmes.

modules de traitement : un POS-Tagger, un processeur sémantique et un générateur de cas de test CSP et de quatre bases de connaissances (lexique, grammaire des cas, ontologie et événements CSP).

Two-level Grammar (TLG) [van Wijngaarden, 1965] est un langage de spécification d'exigence orienté objet. TLG consiste en deux grammaires non contextuelles qui définissent l'ensemble des types du domaine et l'ensemble des fonctions agissant sur ces types. Il est suffisamment formel pour permettre une transformation automatique vers un langage de spécification formel tel que VDM++ [Bryant et Lee, 2002].

Le langage JAPE (*Java Annotation Patterns Engine*) [Cunningham *et al.*, 2000] permet de décrire des patrons d'expressions régulières basés sur les annotations issues du framework GATE. Ce langage est couramment utilisé dans le processus de peuplement d'ontologie pour permettre l'association des résultats d'extraction des éléments de l'ontologie, tels que des EN à des concept [Witte *et al.*, 2010; Ruiz-Martìnez *et al.*, 2011] ou de relations entre EN [Khelif *et al.*, 2007].

La définition de grammaires d'extraction nécessite une connaissance poussée de la forme syntaxique des textes traités et une définition manuelle de patrons à partir de l'étude de styles de rédaction des exigences d'un domaine spécifique, ce qui les rend très dépendantes du style des spécifications choisies. Cette tâche devient ardue quand il s'agit de travailler sur des spécifications issues de styles de rédaction hétérogènes, ce qui est majoritairement le cas lorsque l'on souhaite traiter des spécifications LN rédigées par des utilisateurs non spécialistes. Pour cela, cette approche ne semble pas être en mesure de capturer la sémantique véhiculée par des textes qui, par la nature riche et évolutive du langage naturel n'obéiront que très peu à des grammaires figées.

L'usage de grammaires locales est donc peu fréquent dans les approches de peuplement d'ontologie qui se fondent plus généralement sur la modélisation du contexte linguistique des mentions à identifier dans les textes.

2.3.6.4 *Modélisation du contexte linguistique*

La modélisation du contexte linguistique est fondée sur l'exploitation d'outils et de méthodes du traitement automatique du langage. À la croisée de la linguistique, de l'informatique et de l'intelligence artificielle, le traitement automatique du langage est une discipline qui permet l'analyse des textes en langage naturel. Les outils du TAL servent à l'annotation des textes par des informations lexicales, syntaxiques ou sémantiques. Dans le processus de peuplement d'ontologie, ils sont notamment utilisés pour de l'analyse morpho-lexicale

[Cimiano et Völker, 2005; Tanev et Magnini, 2008; Makki *et al.*, 2009; Faria *et al.*, 2013] qui permet d'identifier la catégorie grammaticale de chaque terme, de la reconnaissance d'entités nommées [Hasegawa *et al.*, 2004; Cimiano et Völker, 2005; Amardeilh, 2006; Magnini *et al.*, 2006; Tanev et Magnini, 2008; Maynard *et al.*, 2008; Ruiz-Martìnez *et al.*, 2011; Faria *et al.*, 2013] qui permet de reconnaître les noms qui réfèrent à des entités uniques du monde, tels que des personnes ou des organisations, de l'identification de co-référence [Faria *et al.*, 2013] qui a pour objectif de lier les noms et pronoms qui réfèrent aux mêmes entités dans les textes ou de l'analyse en dépendances syntaxiques [Lin et Pantel, 2001; Sudo *et al.*, 2001; Nakamura-Delloye, 2011; Nakamura-Delloye et Stern, 2011] qui permet de générer pour chaque phrase les liens syntaxiques entre les termes. Ces informations sont souvent prises comme entrées du processus de reconnaissance de mentions d'instances dans les textes.

On distingue deux types d'approches pour le peuplement d'ontologie : celles qui traitent les termes et celles qui traitent les entités nommées (EN). L'extraction de termes repose en général sur des critères de composition morpho-syntaxique et de récurrence tandis que l'extraction d'entités nommées repose davantage sur des règles typographiques et contextuelles [Mondary *et al.*, 2008]. Dans les deux cas le contexte a une importance particulière. Le contexte correspond aux régularités que l'on peut noter lors de la mention d'un élément linguistique dans les textes. Sa définition est bien souvent fondée sur l'exploitation jointe de résultats d'analyse syntaxique des textes tels que les catégories syntaxiques ou les dépendances syntaxiques et de ressources lexicales permettant la catégorisation linguistique et conceptuelle des termes issus des textes.

Dans le cadre d'une analyse orientée objet de spécifications LN. Abbott [1983] suggère de considérer que les noms représentent des objets candidats et que les verbes représentent des opérations candidates sur eux. Cependant, le langage naturel est un véhicule terriblement imprécis, au sein duquel les noms peuvent être verbalisés et les verbes nominalisés, ce qui par conséquent, peut fausser une liste de candidats objets ou opérations [Booch, 2004]. Aussi, si la définition du contexte d'identification des termes est parfois centrée sur l'usage de catégories syntaxiques (nom, verbe, adjectif...), elle s'accompagne le plus souvent de la considération des dépendances syntaxiques (sujet, objet direct ...) qui les lient. Par exemple, afin de produire une forme structurée et orientée objet, décrite en Two Level Grammar (TLG), d'une spécification initiale en langage naturel , Bryant et Lee [2002], analysent les spécifications et déterminent les objets du domaine à partir des noms issus des spécifications et les interactions entre les objets à partir des verbes et de leurs relations, i.e. les dépendances

syntaxiques [Lee et Bryant, 2002]. Makki *et al.* [2009] partent du principe que les relations sémantiques entre concepts sont le plus souvent représentées par des verbes. Ils extraient des relations issues d'une analyse des verbes et de leur entourage afin d'extraire des triplets, en utilisant des listes de verbes synonymes associés aux relations conceptuelles construites à l'aide de *WordNet*.

Lors d'un processus de peuplement d'ontologie, les termes appartenant à une même classe sémantique ne sont pas nécessairement synonymes. En effet, dans ce cas, les termes doivent être regroupés suivant l'hypothèse Harrissienne [Harris, 1989] selon laquelle le sens des termes peut se déduire de leurs emplois. Le recours à des ressources linguistiques structurellement indépendantes du domaine, tels que *WordNet* ou *VerbNet* n'est alors le plus souvent que peu pertinent.

Afin de représenter le contexte entre paire d'entités nommées (EN), Hasegawa *et al.* [2004] se fondent sur l'hypothèse que les couples d'EN qui apparaissent dans un contexte similaire peuvent être regroupés et que les paires d'EN d'un même groupe sont des instances de la même relation. Dans leur approche, le contexte est représenté par les mots co-occurrents entre deux EN. Dans le cas où les contextes reliant une paire d'EN peuvent dénoter différentes relations, il est attendu que la paire ne soit associée à aucun groupe ou qu'elle soit associée au groupe correspondant à la relation la plus fréquemment exprimée. La limite de cette approche est qu'elle ne permet d'associer qu'une seule relation par paire d'EN.

Une autre approche consiste à considérer les phrases non comme des ensembles de mots linéaires mais sous formes de graphes orientés [Lin et Pantel, 2001; Sudo *et al.*, 2001; Kramdi *et al.*, 2009; Nakamura-Delloye, 2011]. L'analyse syntaxique des phrases d'un texte permet de faire émerger des régularités susceptibles de traduire la présence de relations sémantiques. Plusieurs travaux ont tiré parti de la structure grammaticale de la phrase pour y repérer la mention de propriétés sémantiques. Ces travaux font appel à une représentation sous forme d'arbres de dépendances syntaxiques. L'avantage des arbres de dépendances est qu'ils représentent des dépendances syntaxiques entre des mots pouvant être distants dans une phrase [Fundel *et al.*, 2007]. Par exemple, Kramdi *et al.* [2009] adaptent l'algorithme LP^2 [18] pour l'extraction de relations entre concepts ou instances de concepts, en modifiant la notion de contexte reposant sur une fenêtre de mots par un contexte fondé sur l'exploitation d'arbres de dépendances syntaxiques, qui permettent de repérer aisément les différents arguments d'un verbe dans une phrase.

18. Un algorithme supervisé d'extraction d'instances de concepts qui utilise des informations morpho-syntaxiques, et la reconnaissance d'entité nommé [Ciravegna, 2001]. LP^2 permet d'induire des règles d'extraction à partir de corpus annotés.

Lin et Pantel [2001] considèrent que les propriétés binaires peuvent être représentées par un chemin dans un arbre syntaxique de dépendances et que ces chemins traduisent le même sens, s'ils tendent à se connecter aux mêmes ensembles de mots. Suivant la même intuition, Sudo *et al.* [2001] proposent un modèle à base d'arbres de dépendances pour l'extraction d'informations en japonais ; ce modèle est fondé sur des chemins qui relient les informations dans les arbres de dépendances syntaxiques. Suivant encore la même hypothèse Nakamura-Delloye et Villemonte De La Clergerie [2010] proposent une méthode d'acquisition semi-supervisée pour l'extraction d'entités nommées fondée sur un principe d'induction. L'objectif est de reconnaître à partir de quelques exemples de couples d'entités nommées, les chemins syntaxiques, i.e. les contextes pouvant correspondre à des relations sémantiques, puis d'exploiter ces contextes pour l'identification de nouveaux exemples de couples d'entités. Cette approche est confrontée à la difficulté de déterminer des exemples de départ pertinents correspondant à des relations intéressantes. Nakamura-Delloye et Stern [2011] en proposent une amélioration faisant appel à une méthode non-supervisée qui exploite les informations des dépendances syntaxiques liant les entités nommées reconnues dans les textes. Les paires d'EN sont alors regroupées selon un calcul de similarité lexicale. La limite de cette approche est qu'elle ignore les chemins ne contenant pas de nœuds syntaxiques intermédiaires, où les EN sont directement liées par une seule dépendance syntaxiques. Ces propriétés peuvent dénoter par exemple des relations d'*appartenance* tels que *Est-membre-de* ou *Est-affilié-à*. De façon plus générale, elles correspondent aux propriétés sémantiques dont la mention peut se faire de manière implicite. Selon les auteures, la prise en compte de ces chemins souvent très fréquents mèneraient à la constitution d'un très grand groupe englobant plusieurs sous-groupes tels que les groupes des relations *Est-membre-de* ou *Est-affilié-à*. Dans ce cas, la distinction des différentes relations doit se faire manuellement.

Notre approche diffère de celles citées, en ce sens que notre objectif ne se limite pas à la simple reconnaissance des mentions d'instances de propriétés sémantiques dans les textes et doit aller jusqu'à la reconnaissance précise de chaque type d'instances de propriétés. Il ne s'agit donc pas seulement de reconnaître les contextes pouvant dénoter la présence d'une instance de propriété mais plutôt, de reconnaître les contextes qui dénotent une instance de propriété particulière, i.e. dont la propriété [19] est modélisée dans l'ontologie. Permettre la reconnaissance d'une propriété particulière dans un texte suppose d'inclure des

19. Dans notre approche, chaque propriété est définie entre un domaine et une image. Ces derniers représentent les deux ensembles d'individus pouvant être liés par la propriété.

connaissances la caractérisant dans la définition de son contexte d'apparition dans les textes. Par exemple, dans les deux phrases, *Jean réserve son ticket* et *Jean annule son ticket*, les contextes syntaxiques reliant les termes *Jean* et *ticket* sont similaires. Cependant, chacun d'eux traduit une propriété sémantique différente, respectivement *Réserve(jean, ticket)* et *Annule(jean,ticket)*. Dans ce cas, les distinguer, peut consister à prendre en compte les dénotations possibles de chaque propriété [20] dans les chemins syntaxiques, ce qui reviendrait à considérer non plus des couples mais des triplets de termes. L'inclusion des dénotations possibles de propriétés sémantiques dans la définition de contextes syntaxiques peut de plus, permettre l'identification de chemins syntaxiques récurrents liant des termes dénotant une instance de concept à des termes dénotant une instance de propriété. Cela pourrait autoriser la reconnaissance d'instances de propriétés même lorsque celles-ci contiennent de l'information implicite. Par exemple, à partir de la phrase *Lorsque j'entre dans la cuisine allumer la lumière*, l'instance de propriété *Allume(?i, lumière)* doit pouvoir être extraite même si l'instance du domaine ?i [21] n'est pas explicitée.

Il ressort des travaux de Nakamura-Delloye et Stern [2011], qu'il est important de distinguer les propriétés sémantiques devant être lexicalisées, de celles pouvant être énoncées de manière implicite. Par exemple, distinguer les propriétés *Couleur-voiture* ou *Longueur-trajet* dont les instances peuvent être respectivement dénotées par *voiture rouge* ou *court trajet*, de celles qui doivent nécessairement être explicitées dans les textes telles que *Réserve* ou *Annule*. Cette distinction doit permettre d'autoriser ou d'exclure la recherche de liens directs, c'est-à-dire, qui n'imposent pas la lexicalisation de la propriété, entre les paires de termes dénotant ses domaine et image.

Dans la suite, nous montrons comment la recherche de triplets de termes, i.e. l'inclusion de dénotations d'instances de propriétés dans la définition de contextes syntaxico-sémantiques permet d'une part, de distinguer de façon précise les propriétés sémantiques, diminuant ainsi l'ambiguïté possible et d'autre part, de reconnaître des instances de propriétés même lorsque celles-ci contiennent de l'implicite, et ce dans l'optique d'inférer les connaissances implicites au sein de l'ontologie.

2.3.7 *Classification, identification et validation des individus*

Une fois les mentions d'instances reconnues dans les textes, il s'agit de les classer. La classification d'instances dans l'ontologie concerne

20. ici, *réserve* ou *annule*
21. ?i représente l'individu responsable de l'action d'allumer.

les instances de concepts, soit les individus. Elle consiste à leur associer une classe sémantique, c'est-à-dire affecter chaque individu à un concept. La classification peut reposer sur la reconnaissance de la dénomination du concept ou sur la reconnaissance des mentions de ses propriétés définissantes. Les deux posent le problème de mentions pouvant être ambiguës.

La classification d'individus peut se faire de deux manières : au sein de l'ontologie ou hors de l'ontologie. Dans le premier cas, il s'agit de faire usage des axiomes terminologiques de l'ontologie pour inférer les classes d'appartenance sémantique de chaque individu. Dans le second cas, les classes sémantiques des individus sont déterminées avant leur création dans l'ontologie, ce qui fait appel à l'exploitation des connaissances issues de l'analyse des textes. La majorité des approches de peuplement d'ontologie s'orientent vers cette seconde solution [Giuliano et Gliozzo, 2008; Makki *et al.*, 2009; Ruiz-Martìnez *et al.*, 2011; Thongkrau et Lalitrojwong, 2012; Anantharangachar *et al.*, 2013; Sun *et al.*, 2013] et utilisent un ensemble d'heuristiques et de patrons issus de techniques d'extraction d'information pour identifier les classes sémantiques dénotées par les termes du texte [Makki *et al.*, 2009; Anantharangachar *et al.*, 2013] ou des approches faisant appel à de l'apprentissage automatique [Giuliano et Gliozzo, 2008; Thongkrau et Lalitrojwong, 2012; Sun *et al.*, 2013].

À notre connaissance, seul BOEMIE [Castano *et al.*, 2008; Petasis *et al.*, 2013] s'intéresse à l'utilisation des mécanismes d'inférences de l'ontologie pour classer les instances de concepts. BOEMIE n'extrait que des instances de concepts qu'il considère comme primitifs, i.e. qui ne sont pas composés d'autres concepts (par opposition aux concepts complexes). Par la suite, il applique un raisonnement sur les propriétés portant sur les instances de concepts primitifs pour classer les instances de concepts complexes.

Certaines instances extraites peuvent être préalablement connues dans l'ontologie sous un autre nom, lorsque le langage d'ontologie ne fait pas la supposition du nom unique (tel que OWL). Une fois les instances classées, il peut donc être utile, comme cela est pointé dans [Takhirov *et al.*, 2012], de les lier aux entités identiques. Cela consiste à identifier au sein de l'ontologie, les instances qui réfèrent à la même entité. Cet aspect du peuplement est assez peu abordé dans la littérature. Néanmoins, la création de nouvelles instances sans vérifier si l'objet réel qu'elles dénotent existe déjà, peut mener à la création d'instances redondantes, qui si elles contiennent des informations contradictoires, devraient mener à une ontologie incohérente ou à représenter une interprétation du monde erronée ou encore fausser l'interprétation du monde ; l'interprétation d'un texte est totalement

différente selon que l'on considère que deux expressions ou termes du texte réfèrent ou non à une même entité [Gayral *et al.*, 1996].

Le système *SOBA* [Buitelaar et Siegel, 2006] stocke des instances extraites dans une base de connaissances (l'ontologie) et utilise ensuite l'ontologie pour interpréter et lier les instances nouvellement extraites par rapport à celles déjà existantes. À l'aide de requêtes pour interroger l'ontologie avant la création de nouvelles instances, il lui est possible de réutiliser les instances existantes qui référeraient au même objet réel que les nouvelles instances à créer. Maynard *et al.* [2007] proposent un solution au problème de résolution d'identité et de fusion d'instance qui fonctionne en quatre étapes : pour chaque nouvelle instance candidate, un ensemble d'instances (du même concept par exemple) est extrait de la base de connaissances. Deuxièmement, les connaissances de chaque instance candidate sont recueillies (par exemple, les attributs et les valeurs stockées dans la base de connaissances). Enfin, une décision est prise sur la base de la similitude entre le nouvel individu et les individus existants. Cette décision est fondée sur un ensemble de règles définies par l'expert du domaine qui sont utilisées pour calculer un score de similarité entre le nouvel individu et chaque individu existant (ces règles peuvent par exemple vérifier le nom d'emprunt, ou la similitude entre les valeurs des attributs similaires). Le système OntoPop [Amardeilh, 2007] aborde la question de consolidation des connaissances avant l'instanciation de l'ontologie. OntoPop intègre un module de consolidation qui s'applique en trois étapes. Premièrement, il vérifie à l'aide de requêtes sur la base de connaissances que l'entité extraite n'existe pas déjà dans l'ontologie. Deuxièmement, il vérifie que l'entité extraite est cohérente par rapport à la classe sémantique à laquelle elle appartient (valeurs d'attributs, cardinalités ...). La dernière étape est optionnelle, elle consiste à se connecter à un moteur de raisonnement pour inférer de nouvelles entités en relation et contrôler la cohérence globale de l'ontologie après ajout des nouvelles connaissances. Dans cette approche, les triplets (instances de propriétés) invalides sont annotés et stockés hors de l'ontologie afin d'être présentés ultérieurement à un utilisateur final pour une validation manuelle. Le système Artequakt [Kim *et al.*, 2002; Alani *et al.*, 2003] s'intéresse à la consolidation de connaissances issues du Web. Il est fondé sur la comparaison et la fusion des différentes individus (par exemple, les lieux, les personnes, les dates) de l'ontologie. *Artequakt* applique un ensemble d'heuristiques définies manuellement et des méthodes de raisonnement dans le but de détecter automatiquement des contradictions et d'identifier et de fusionner les connaissances dupliquées (individus ou valeurs de propriétés) dans l'ontologie. Le système *BOEMIE* [Castano *et al.*, 2008; Petasis *et al.*, 2013] améliore l'approche d'*Artequakt* par l'utili-

sation de techniques d'apprentissage automatique au lieu d'heuristiques manuellement développées. *BOEMIE* mesure la similarité de deux individus en fonction des propriétés qu'ils partagent, en comparant les assertions sans en distinguer pour autant de pertinentes pour l'identification d'individus. Ces similarités sont utilisées dans le domaine de l'athlétisme afin de trouver des athlètes ou des événements sportifs similaires. Cette approche a comme inconvénient de ne pas définir les propriétés ou valeurs de propriétés les plus discriminantes pour l'identification d'un individu comme on peut le faire dans le domaine des bases de données à l'aide de clés primaires identifiant de façon unique les objets réels du monde, cette approche peut donc amener à considérer des individus comme similaires sur la base de propriétés non discriminantes. Selon de Melo [2013] il n'existe aucun moyen conventionnel de choisir les propriétés discriminantes pour déterminer la quasi-identité et la similarité entre individus. D'après nous, ces propriétés sont typiquement celles qui permettent de définir les conditions d'identité entre les individus de chaque concept individuel. La question qui se pose alors est : quelles propriétés distinguent un individu d'un autre ?

There is no universal agreed-upon way of determining which properties should count as salient in determining near-identity and similarity.

Dans la plupart des cas, il est plus important d'avoir une ontologie précise et correcte plutôt qu'une ontologie surpeuplée d'instances incorrectes [Celjuska *et al.*, 2004]. Deux points cruciaux sont alors à valider pour chaque nouvelle instance : l'instance candidate est-elle véritablement un membre des classes sémantiques auxquelles elle est associée ? Et l'instance candidate correspond-elle véritablement à l'entité à laquelle elle est associée ?

Pour assurer et maintenir la cohérence de l'ontologie, il est important de vérifier la validité des instances extraites automatiquement des textes. La validation d'instances de classe peut être abordée de différentes manières, telles que par le calcul de la co-occurrence entre instances (candidates et validées) [Kozareva *et al.*, 2008; Wang et Cohen, 2009], par de l'apprentissage [Pang *et al.*, 2009; Takhirov *et al.*, 2012] ou encore de manière supervisée comme dans l'outil OntoSophie [Celjuska et Vargas-vera, 2004] qui prend l'utilisateur à partie pour accepter, refuser ou modifier les nouvelles instances extraites. Pang *et al.* [2009] proposent une approche de validation d'instances qui repose sur l'usage des caractéristiques du concept. Takhirov *et al.* [2012] vérifient la validité des entités extraites et des relations auxquelles elles participent par des méthodes d'apprentissage. Ils font appel à un classifieur entraîné sur des ensembles d'exemples pour calculer la validité de la classe sémantique à laquelle sont associées les entités. Ruiz-Martìnez *et al.* [2011] considèrent les entités nommées reconnues dans les textes comme de potentielles instances candidates. En cas d'annotations ambiguës (i.e. plusieurs annotations possibles),

un arbre de désambiguïsation est créé. Chacune des branches de cet arbre contient l'un des groupes d'annotations possibles. Un score est alors affecté à chaque groupe. Le groupe possédant le plus haut score est retenu pour peupler l'ontologie. En dernier, lieu un raisonnement ontologique est appliqué afin de vérifier la cohérence de l'ensemble de l'ontologie.

L'ontologie initiale du processus de peuplement est supposée consistante. La vérification de l'ontologie nouvellement peuplée peut mener à la découverte d'incohérences. Dans ce cas l'ajout de nouvelles instances est en cause. Néanmoins cela ne veut pas forcément dire que ces nouvelles instances sont défectueuses. Ces nouvelles instances peuvent révéler la présence d'instances non conformes. Dans tous les cas, il peut s'avérer utile de conserver un lien entre les instances défectueuses dans l'ontologie et leurs mentions dans les spécifications. À l'exemple, des opérations de consolidation proposées dans le travail de thèse d'Amardeilh [2007] qui produisent deux réseaux sémantiques. Le premier conserve la connaissance contrôlée et validée. Le second représente la connaissance qui a été rejetée. Il est conservé pour une validation humaine ultérieure. Conserver la trace des connaissances défectueuses doit permettre de comprendre la cause des incohérences à savoir, soit les spécifications concernées sont incorrectes et doivent être corrigées, soit l'extracteur d'instances s'est trompé et il faut l'améliorer, soit l'ontologie a un problème de conceptualisation.

Il apparaît clairement que le formalisme logique de l'ontologie et les possibilités de raisonnement qu'il offre sont peu exploités dans le processus de classification, d'identification et de validation des individus d'une ontologie issus d'une analyse de textes. Pourtant les raisonneurs ontologiques sont en général construits pour l'application de ces trois tâches. Dans notre approche, nous mettons à profit le formalisme logique de l'ontologie pour spécifier de manière formelle les propriétés définissantes des concepts et des individus de l'ontologie qui permettront la classification, l'identification et la validation des individus au sein de l'ontologie.

2.4 CONCLUSION

Le peuplement d'ontologie est une discipline assez récente pour laquelle plusieurs questions fondamentales restent ouvertes. Dans cette section, nous avons présenté et discuté un certain nombre d'approches pour le peuplement d'ontologie. Cette discussion s'est déroulée en fonction des différentes parties qui composent le processus de peuplement d'ontologie. Il en ressort plusieurs observations : a) l'ontologie initiale est au cœur du processus de peuplement, cepen-

dant sa place n'a pas été suffisamment discutée dans la littérature ; b) peu d'approches s'intéressent à l'usage du formalisme logique des ontologies et des possibilités de raisonnement offertes lorsque l'engagement sémantique dans la formalisation de l'ontologie est élevé ; c) plusieurs méthodes de peuplement exploitent les instances initiales de l'ontologie pour l'identification du même type d'instance ou d'un autre ; néanmoins la question de la précédence dans l'identification d'un type d'instances (individus, instances de propriété) par rapport à un autre est assez peu discuté. Il semble pourtant bien y avoir une précédence nécessaire entre l'identification d'instances de propriétés et celles d'individus que nous avons nommé référents individuels, i.e. représentant des entités du monde qui ne sont identifiables que par leurs valeurs de propriétés. Nous avançons donc qu'il est plus pertinent de guider le peuplement d'une ontologie par l'identification d'instances de propriétés à partir des textes. Puis, de classer, d'identifier et de valider les individus de l'ontologie à l'aide d'un raisonnement ontologique.

3

DES SPÉCIFICATIONS EN LANGAGE NATUREL AU MODÈLE ONTOLOGIQUE

Sommaire

« L'accumulation des connaissances n'est pas la connaissance. »

Une Histoire de la lecture. Alberto Manguel

« Beaucoup de réflexion et non beaucoup de connaissances, voilà à quoi il faut tendre. »

Démocrite

INTRODUCTION

Nous rappelons que l'objectif de ce travail est l'analyse automatique de spécifications d'exigences en langage naturel, via la formalisation et l'application de méthodes formelles. Les spécifications ciblées décrivent les règles de comportement d'un système. Atteindre cet objectif suppose d'être en mesure de fournir une représentation des spécifications suffisamment formelle, pour autoriser un passage vers un langage de spécification formelle. Nous avons choisi de traiter ce problème de passage de l'informel au formel en deux temps, en passant par l'intermédiaire d'une ontologie. Ce chapitre porte sur la première phase qui consiste à passer des spécifications en langage naturel à leur représentation ontologique. La seconde phase qui consiste à passer du modèle ontologique au langage de spécification formelle est décrite et discutée au chapitre 4.

Dans ce chapitre, nous décrivons nos choix de modélisation pour représenter le comportement générique d'un système, les vérifications permises par OWL ainsi que notre approche de peuplement d'ontologie visant à spécialiser le comportement générique modélisé, en fonction des spécifications en langage naturel.

La structure de ce chapitre est la suivante : en section 3.1, nous décrivons nos choix de modélisation et proposons une structure générique pour une ontologie du comportement d'un système. En section 3.2, nous détaillons les vérifications que nous souhaitons effectuer et décrivons celles possibles sous OWL. Enfin, nous présentons notre approche de peuplement automatique d'ontologie mettant à profit les caractéristiques de notre modélisation et les mécanismes de raisonnement offerts par OWL.

3.1 MODÉLISATION DU COMPORTEMENT D'UN SYSTÈME

Notre approche de transformation de spécifications en langage naturel suppose l'existence d'une ontologie formelle modélisant le comportement d'un système particulier. L'ontologie a pour vocation de

définir un cadre formel suffisant pour guider l'analyse de spécifications d'exigences. Ce cadre doit permettre de :

1. définir les composants du système, leurs caractéristiques et leurs règles de comportement ;
2. permettre la reconnaissance, à partir des spécifications d'exigences, des mentions faisant référence aux composants du système et aux règles de comportement.

Cette section a pour objectif de définir la forme et les caractéristiques générales d'une telle ontologie. Pour cela, nous donnons les lignes directrices permettant de définir un modèle ontologique permettant une analyse fine et adéquate des spécifications.

3.1.1 *Choix de conceptualisation*

Selon Gruber [1993], une conceptualisation est une vue abstraite et simplifiée du monde que nous voulons représenter dans un but particulier. Dans cette section, nous décrivons notre point de vue sur le monde représentant le comportement d'un système et motivons nos choix de conceptualisation.

3.1.1.1 *Concepts et propriétés*

L'ontologie que nous souhaitons modéliser doit comporter les concepts et propriétés clés du comportement d'un système. Les concepts et propriétés représentent la structure conceptuelle de l'ontologie. Cette structure modélise un aspect général du domaine de connaissance.

L'ontologie doit définir des concepts et propriétés suffisamment génériques pour représenter l'ensemble des types de composants d'un système et définir des concepts et propriétés assez spécifiques pour permettre la distinction des différents composants. Elle n'a pas vocation à représenter de façon exhaustive l'ensemble du domaine de spécialité. Elle doit être pensée pour tirer profit de son potentiel d'inférence, et non seulement comme un moyen d'accumuler et de structurer les connaissances. Nous ne faisons pas d'hypothèse sur la profondeur hiérarchique de l'ontologie, ce qui importe est d'expliciter pour chaque concept les propriétés impliquées dans les conditions nécessaires et suffisantes de classification et d'identification de ses individus, cela à l'aide d'axiomes terminologiques. Toute définition de nouveaux sous-concepts doit s'accompagner de la définition des conditions de classification et d'identification de ses individus.

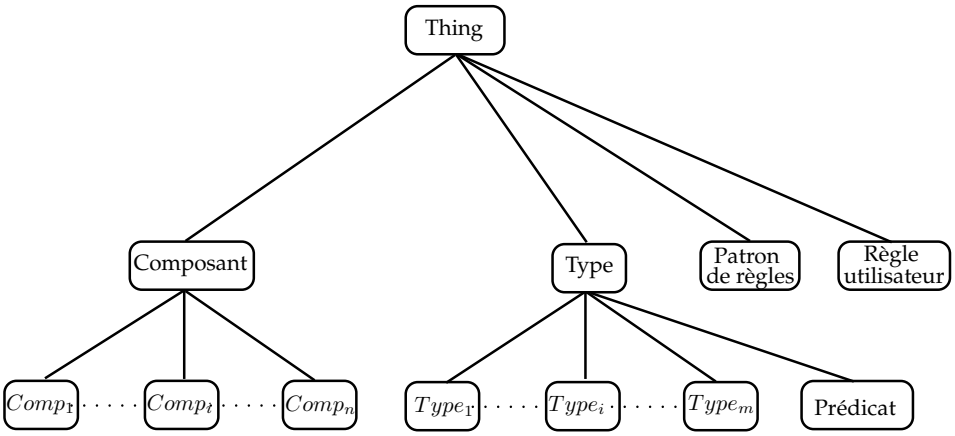


FIGURE 10: Structure générique du comportement d'un système

Notre première proposition distingue deux sortes de concepts : *Composant* et *Type* [1] (cf. figure 10). 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 (composants physiques, composants logiciels, phénomènes, etc.) ;
2. Chaque sous-concept de *Type* représente un ensemble de types spécifiques au domaine. Les individus *Type* étendent les types prédéfinis simples (entier, réel, booléen, chaîne de caractère ...) par des types propres au domaine, permettant de caractériser les composants du système. Ces types peuvent par exemple correspondre à une couleur, une marque, un modèle ...

Notre seconde proposition est la distinction entre deux types de propriétés :

1. Les propriétés qui décrivent une interaction entre deux composants du système. Elles sont définies sur deux ensembles de concepts, sous-concepts de *Composant*. Par exemple, une instance de propriété telle que *Actionne*(i_{C_x},i_{C_y}) décrit une *relation* transitoire entre les deux individus i_{C_x} et i_{C_y} ;
2. Les propriétés qui décrivent les caractéristiques des composants, définies entre un ensemble de concepts, sous-concepts de *Composant* et un ensemble de concepts, sous-concepts de *Composant*,

1. Le concept *Prédicat* sous-concept de *Type* (cf. figure 10) est décrit ultérieurement.

de *Type* ou un ensemble de type simple. Par exemple, une instance de propriété telle que *Fixé-à*(i_{C_x},i_{C_y}) caractérise i_{C_x} par la valeur d'*attribut* i_{C_y}.

Cette distinction mène à la redéfinition des deux types de propriétés *Relation* et *Attribut*.

DEFINITION 12: une **propriété de type relation** est une sous-propriété de la super-propriété *Relation*. Sous OWL elle correspond à une *ObjectProperty* (cf. Figure 11) définie entre deux sous-concepts *Composant*.

DEFINITION 13: une **propriété de type attribut** est soit une sous-propriété de la super-propriété, *Attribut* qui sous OWL correspond à une *ObjectProperty* (cf. Figure 11) définie entre un sous-concept de *Composant* ou de *Type*[2] et un sous-concept de *Composant* ou de *Type*, soit une propriété de type *DataProperty* (cf. Figure 11) définie entre un sous-concept de *Composant* ou de *Type* et un type simple OWL.

On distingue également deux types d'attributs. Les attributs *dynamiques* dont la valeur est amenée à évoluer au cours du temps, tel que le solde d'un compte en banque et les attributs *statiques* dont la valeur n'est pas amenée à évoluer, tel qu'un identifiant de compte bancaire. Ce dernier type d'attribut correspond aux propriétés *définissantes* d'un concept qui peuvent permettre d'identifier et de distinguer ses individus. Au sein de l'ontologie, cette distinction entre attribut est marquée par les labels *dynamic* et *static*.

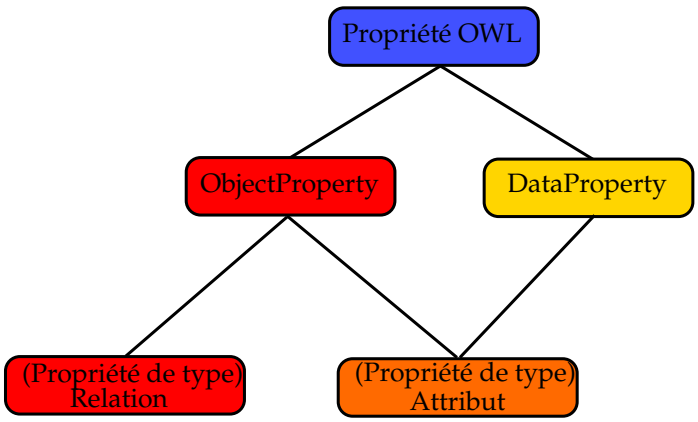


FIGURE 11: Structure générale des propriétés de l'ontologie

3.1.1.2 *Individus*

Les individus des sous-concepts et du concept *Composant* correspondent à toutes les entités qui composent le système et participent

2. Un individu de *Type* peut être défini par d'autres individus de *Type*. Par exemple, l'individu *twingo* a comme valeur de marque *renault*.

à son comportement. Les individus des sous-concepts et du concept *Type* correspondent aux types spécifiques au domaine, non prédéfinis par le langage de représentation d'ontologie (en l'occurrence OWL).

Si l'ontologie représente un modèle abstrait, le système représenté lui, est bien réel. Les individus composant le système correspondent à des objets réels du monde. En revanche, les individus du concept *Type* représentent des notions abstraites dont le rôle est de caractériser les composants. L'existence des individus du concept *Type* ne dépend donc pas des spécifications utilisateur car ils font partie de la définition du domaine. Il en est de même de l'existence de certains individus du concept *Composant* qui décrivent la configuration physique du système. Ces individus particuliers doivent être rattachés aux objets réels qu'ils représentent, afin de pouvoir faire le lien entre conceptuel et réel. Pour cela, nous définissons dans notre modèle la super-propriété *Num-identifiant*. Une propriété, sous-propriété de *Num-identifiant*, est créée pour chaque individu associé à un composant physique du système. Elle peut par exemple correspondre au *numéro de série* d'un objet, à l'immatriculation d'une voiture ou encore à l'*URI* d'un capteur. *Num-identifiant* a pour objectif d'identifier de manière unique les objets du monde réel liés aux individus conceptuels. Dans notre conceptualisation, cette propriété n'est pas de type *fonctionnelle*, i.e. différents individus peuvent posséder le même numéro identifiant. Ceci permet de modéliser les individus composants du système, non pas à partir des objets réels qu'ils représentent mais à partir du comportement qui les caractérise. Par exemple, si l'on considère la conceptualisation d'un réseau de capteurs, le choix technique d'installer dans une zone définie un seul capteur pouvant détecter plusieurs types de phénomènes ou plusieurs capteurs pouvant détecter chacun un type de phénomènes n'aura pas d'importance, car les possibilités de configuration de leur comportement seront identiques. Dans les deux cas, notre modèle représentera pour la zone définie autant de capteurs conceptuels que de types de phénomènes qu'ils prennent en charge. La différence sera faite par le numéro identifiant : dans le premier cas, il sera le même pour l'ensemble des capteurs conceptuels et sera différent pour chacun d'eux dans le second.

Les individus peuvent participer à deux types d'instances de propriété : les relations et les attributs. La figure 12 illustre le principe d'association. Dans cette figure, le niveau conceptuel est représenté par le plan du haut qui représente les concepts de l'ontologie et le plan du bas qui représente les propriétés de l'ontologie. Le niveau assertionnel correspond au plan du milieu. Il représente les individus de l'ontologie. Les fonctions ($\mathfrak{I}^C$) qui associent les concepts à leurs individus sont représentées par des flèches hachurées, en rouge pour les individus du concept *Composant* et en jaune pour les individus

du concept *Type*. Les fonctions ($\mathfrak{I}^{\mathbb{P}}$) qui associent à chaque propriété un ensemble de couples d'individus sont représentées par des flèches pointillées, en rouge pour les relations et en orange pour les attributs.

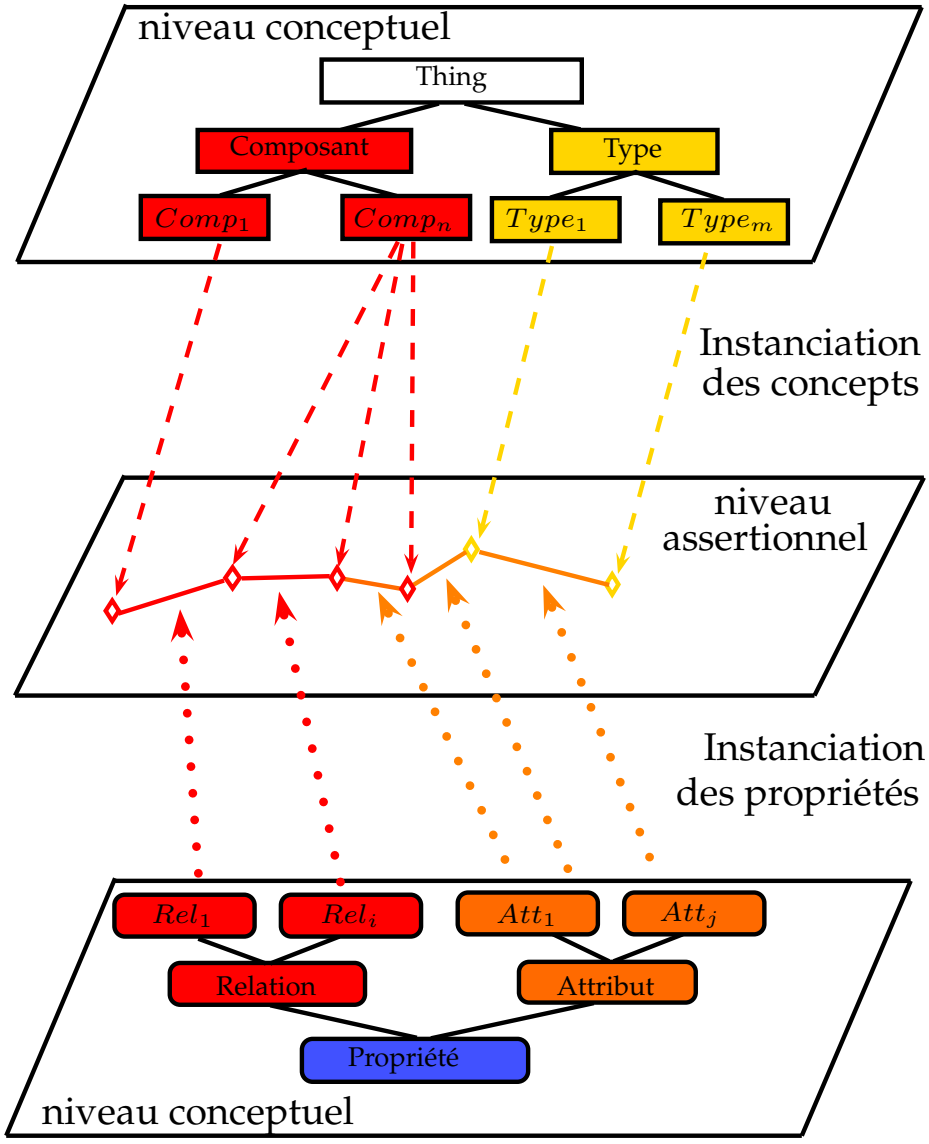


FIGURE 12: Instanciation des concepts et propriétés

3.1.1.3 *Conditions de classification et d'identification des individus*

Nous définissons pour chaque concept de l'ontologie, les conditions nécessaires et suffisantes à la classification de ses individus. Pour les concepts individuels nous définissons en plus les conditions d'identification de leurs individus. L'ensemble de ces conditions est défini à partir des propriétés et de leurs valeurs. Plus précisément, les conditions de classification (cf . section 2.1.4.1) sont définies sur :

- les valeurs du domaine et de l'image des propriétés ;
- les conditions nécessaires et suffisantes définissant des classes d'équivalence.

Les conditions d'identification (cf . section 2.1.4.2) sont définies à l'aide :

- de contraintes fonctionnelles ou inverses fonctionnelles sur les propriétés ;
- de règles d'inférence SWRL permettant de définir les valeurs de propriétés que deux individus doivent partager pour être inférés comme identiques.

3.1.1.4 *Règles de comportement*

L'objectif de l'analyse des spécifications est de reconnaître et de formaliser les règles de comportement spécifiées par l'utilisateur du système. Ces règles sont dans un premier temps modélisées dans l'ontologie dans l'optique future de les traduire dans un langage de spécifications formelles. Les règles de comportement modélisent l'aspect dynamique du système. Dans notre approche, nous distinguons deux types de règles de comportement : (1) les règles qui décrivent le comportement général que doit adopter le système indépendamment de ses utilisateurs et (2) les règles spécifiques qui décrivent le comportement du système, en fonction des exigences de l'utilisateur. C'est cette deuxième catégorie de règles qui nous intéresse. La première catégorie de règles est définie par le concepteur de l'ontologie afin de guider l'identification de la seconde.

DEFINITION 14: Nous définissons une **règle de comportement** comme une règle d'inférence, constituée d'un *antécédent* et d'un *conséquent*. L'*antécédent* définit les conditions d'application de la règle et le *conséquent* définit le résultat de son application. On note $\mathtt{antécédent} \Rightarrow \mathtt{conséquent}$.

Chacun des *antécédent* et *conséquent* est une conjonction de prédicats. Un prédicat est soit :

- unaire, de la forme $C(i_x)$. Il représente une instance de concept où $i_x \in \mathfrak{I}^C(C)$,
- binaire, de la forme $P(i_x, i_y)$. Il représente une instance de propriété où $(i_x, i_y) \in \mathfrak{I}^{\mathbb{P}}(P)$, où i_x est une instance de concept, i_y est une instance de concept, un littéral (valeur de type simple) ou une variable représentant un individu ou un littéral.

Le conséquent quant à lui, n'est constitué que de prédicats binaires. Dans une ontologie, ces prédicats binaires correspondent à de nouvelles instances de propriétés entre les individus, littéraux ou variables définis dans l'antécédent. Notre approche d'identification de règles utilisateur est guidée par l'identification d'instances de pro-

priété, nous ne considérons donc que les prédicats binaires (instances de propriétés) dans la définition des règles de comportement. En utilisant le vocabulaire de l'ontologie, nous pouvons modéliser une règle de comportement de la manière suivante :

$$P_1(i_j, i_x) \wedge ... \wedge P_m(i_y, i_n) \Rightarrow P_k(i_j, i_y) \wedge ... \wedge P_z(i_x, i_n)$$

avec P_k un prédicat binaire faisant référence à une instance de propriété définie dans l'ontologie et i_x un individu, un littéral (valeur de type simple) ou une variable.

Notre première intuition fut de modéliser les règles de comportement dans l'ontologie à l'aide de règles SWRL, car elles sont formées à partir de prédicats représentant les concepts et propriétés de l'ontologie. Néanmoins, cette solution s'est avérée limitée de par la nature du raisonnement appliqué aux règles SWRL. Lors de ce raisonnement, toutes les règles SWRL de l'ontologie sont appliquées sans tenir compte des aspects séquentiels ou concurrents entre les règles. Cela peut mener à l'application simultanée de règles qui dans la réalité ne peuvent s'appliquer simultanément. Par exemple, soit les deux règles $A \Rightarrow B$ et $A' \Rightarrow B'$ avec A et A' (respectivement B et B') deux antécédents (respectivement deux conséquents) opposés. Deux antécédents (respectivement deux conséquents) sont opposés si ils contiennent des prédicats opposés, i.e. qui représentent des propriétés définies comme opposées dans l'ontologie, telles que *Turn-on* et *Turn-off*. Lors d'un raisonnement sous OWL ces deux règles pourront éventuellement s'exécuter en même temps et donc créer simultanément les deux conséquents B et B' qui auront pour effet de générer une incohérence. Cependant, en pratique elles peuvent coexister si elles ne s'appliquent pas en même temps. Le choix des règles SWRL pour modéliser l'ensemble des exigences a donc été abandonné.

Nous nous sommes donc orientés vers une autre modélisation qui permet de préserver les avantages du formalisme SWRL en terme de représentation sans pour autant être soumis aux mêmes contraintes de raisonnement. Pour cela, nous proposons de modéliser dans l'ontologie deux concepts *Patron-de-Règle* et *Règle-utilisateur*.

Le rôle du concept *Patron-de-Règle* est de représenter les types de règles à rechercher dans les spécifications d'exigences. Il représente l'ensemble des patrons de règles de comportement. Ses individus (les patrons de règles) sont définis par le concepteur de l'ontologie. Le rôle du concept *Règle-utilisateur* est de représenter les règles de comportement spécifiées par l'utilisateur. Ses individus (les règles utilisateur) seront créés automatiquement à partir de l'analyse des spécifications d'exigences. Chacun de ces concepts possède les propriétés *Antécédent* et *Conséquent*. Ces deux propriétés ont comme valeur d'image un prédicat (qui est défini de façon similaire aux prédicats utilisés dans

SWRL). Le type prédicat, n'étant pas défini sous OWL, nous avons défini un concept *Prédicat* sous-concept du concept *Type* (cf. figure 10). Ce concept possède les quatre propriétés *Nom-Prédicat* (nom de concept ou de propriété), *Type-Prédicat* (unaire ou binaire), *Argument-1* (individu, valeur de type simple ou variable) et *Argument-2* (individu, valeur de type simple ou variable). Par exemple, le prédicat $P(i_x, i_y)$ sera représenté par une instance du concept *Prédicat* qui possède les valeurs : P pour la propriété *Nom-Prédicat*, binaire pour la propriété *Type-Prédicat*, i_x pour la propriété *Argument-1* et i_y pour la propriété *Argument-2*.

Une instance du concept *Patron-de-Règle* ou *Règle-utilisateur* peut avoir autant de valeurs pour la propriété *Antécédent* (resp. *Conséquent*) que d'instances du concept *Prédicat* nécessaires pour représenter l'antécédent (resp. le conséquent) qu'elle modélise. Les propriétés définies dans le conséquent d'un patron de règles représentent des super-propriétés. Ces propriétés définissent un type de propriétés du système, telles que des propriétés permettant d'actionner un appareil : *Turn-on*, *Turn-off*, *Increase*, *Decrease*. L'utilisation de super-propriétés permet de définir le type de propriétés attendues dans la partie conséquent d'une règle utilisateur sans avoir à les énumérer.

Dans les instances de patrons de règle, nous nommons *prédicat spécifique*, les prédicats qui contiennent au moins une variable ou qui appartiennent au conséquent de la règle, i.e. qui représentent une super-propriété. Ces prédicats sont ceux qui seront spécialisés à partir de l'analyse des spécifications pour créer des instances de *Règle-utilisateur*. 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*. Elles sont créées durant le processus de peuplement de l'ontologie. La création d'une règle utilisateur consiste en premier lieu à instancier les variables d'une instance de *Patron-de-Règle* par les valeurs issues de l'analyse des spécifications ; en second lieu elle définit les propriétés du conséquent qui seront créées dynamiquement. À chaque instance de règle de comportement créée est associé le numéro de phrase, dont la règle est issue, à l'aide de la propriété *Num-phrase*. Cela permet de garder le lien entre les spécifications textuelles et les règles de comportement créées.

L'instanciation d'un individu de *Règle-utilisateur* est guidée par un individu du concept *Patron-de-Règle*. La figure 13 illustre le principe de spécialisation (d'association) d'une instance de *Patron-Règle* (illustrée à gauche) pour la création d'une instance de *Règle-Utilisateur* (illustrée à droite). Le principe est simple ; les éléments à identifier à partir de l'analyse des spécifications sont définis par les instances de *Patron-Règle*. Ces éléments correspondent à certains arguments des

prédicats spécifiques définis par les propriétés *Antécédent* et *Conséquent* et à l'ensemble des noms de prédicats spécifiques définis par la propriété *Conséquent*. L'ensemble des éléments à identifier sont représentés en bleu et donnent lieu à l'identification d'individus ou littéraux (i_x) ou de noms de propriété i_{P_z}. Dans cette figure, les $?v_i$ correspondent à des variables (de type individu ou littéral) et les c_i à des constantes (individus ou littéraux) que l'on ne souhaite pas instancier à partir de l'analyse des spécification d'exigences.

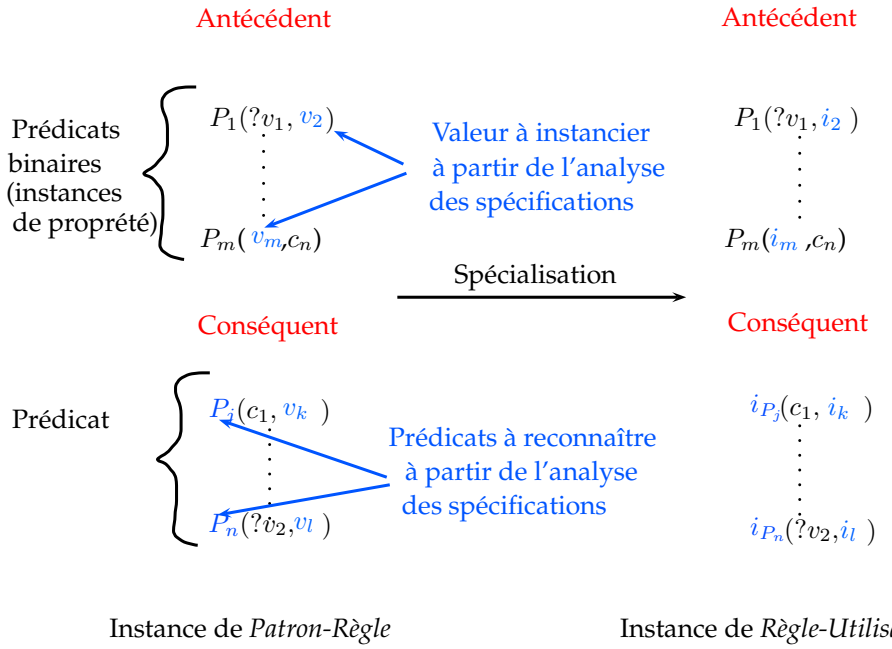


FIGURE 13: Spécialisation d'une instance de *Patron-Règle* en une instance de *Règle-Utilisateur*

3.1.2 *Définition formelle et notation*

Dans la section précédente, nous avons défini une ontologie comme un tuple $< \mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}} >$. Dans cette section, nous présentons les spécificités que l'on doit apporter à cette définition pour définir une ontologie spécifique au comportement d'un système. Nous définissons une ontologie du comportement d'un système de la manière suivante :

$\mathbb{O} =< \mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}} >$ avec

DEFINITION 15: L'ensemble des concepts $\mathbb{C}$ est une partition, contenant quatre ensembles de concepts et sous-concepts, définie de la manière suivante :

$$\mathbb{C} = \mathbb{C}_C \cup \mathbb{C}_T \cup \mathbb{C}_{PR} \cup \mathbb{C}_{RU}$$

- $\mathbb{C}_C$: ensemble de sous-concepts de *Composant* tel que
$$c \in \mathbb{C}_C \equiv c \in \textit{Composant}$$
- $\mathbb{C}_T$: ensemble de sous-concepts de *Type* tel que
$$c \in \mathbb{C}_T \equiv c \in \textit{Type}$$
- $\mathbb{C}_{PR}$: ensemble de sous-concepts de *Patron-Règle* tel que
$$c \in \mathbb{C}_{PR} \equiv c \in \textit{Patron-Règle}$$
- $\mathbb{C}_{RU}$: ensemble de sous-concepts de *Règle-Utilisateur* tel que
$$c \in \mathbb{C}_{RU} \equiv c \in \textit{Règle-Utilisateur}$$

.

DEFINITION 16: L'ensemble des propriétés $\mathbb{P}$ est toujours défini tel que $\mathbb{P} = \mathbb{P}_R \cup \mathbb{P}_A$. Cependant nous modifions légèrement la définition de ces deux sous-ensembles comme suit :

- $\mathbb{P}_R$: l'ensemble des relations P définies entre les sous-concepts de *Composant* tel que $D \triangleleft P \triangleright R$ avec $D \subseteq \mathbb{C}_C$ et $R \subseteq \mathbb{C}_C$ et $\mathfrak{I}^{\mathbb{P}}(P) \subseteq \mathfrak{I}^{\mathbb{C}}[\mathbb{C}_C] \times \mathfrak{I}^{\mathbb{C}}[\mathbb{C}_C]$ [3].
- $\mathbb{P}_A$: l'ensemble des attributs P qui ont pour domaine un sous-ensemble du concept *Composant* ou un sous-ensemble du concept *Type* et qui ont pour image un sous-ensemble du concept *Composant*, du concept *Type* ou un sous-ensemble de types simples (entiers, chaînes de caractères ...) tel que $D \triangleleft P \triangleright R$ avec $D \subseteq \mathbb{C}_C \cup \mathbb{C}_T$ et $R \subseteq \mathbb{C}_C \cup \mathbb{C}_T \cup \mathbb{T}$ et $\mathfrak{I}^{\mathbb{P}}(P) \subseteq (\mathfrak{I}^{\mathbb{C}}[\mathbb{C}_C] \cup \mathfrak{I}^{\mathbb{C}}[\mathbb{C}_T]) \times (\mathfrak{I}^{\mathbb{C}}[\mathbb{C}_C] \cup \mathfrak{I}^{\mathbb{C}}[\mathbb{C}_T] \cup \mathbb{V})$ avec $\mathbb{T}$ l'ensemble de types simples et $\mathbb{V}$ l'ensemble des valeurs de types simples.

3.1.3 *Manipulation de l'ontologie*

Afin de pouvoir manipuler et raisonner sur les ensembles définis par l'ontologie, nous avons développé des fonctions écrites en *Java* en utilisant l'API Java *OWL-API* [4]. *OWL-API* [Horridge et Bechhofer, 2009] est l'implémentation de référence pour créer et manipuler des ontologies OWL. Sa dernière version est centrée sur OWL 2. Nous faisons aussi appel à *Jess* un environnement de moteur de règles et de scripts permettant d'exécuter des requêtes *SQWRL*. Le tableau 1 décrit certaines des fonctions implémentées pour la manipulation des ensembles ontologiques.

3. $\mathfrak{I}^{\mathbb{C}}[\mathbb{C}_C]$ représente l'ensemble des images par $\mathfrak{I}^{\mathbb{C}}$ de tous les éléments de $\mathbb{C}_C$ ($\mathfrak{I}^{\mathbb{C}}[\mathbb{C}_C] = \bigcup_{c \in \mathbb{C}} \mathfrak{I}^{\mathbb{C}}(C)$).

4. http ://owlapi.sourceforge.net/

Fonctions	**Description**	**Résultat**
getOntologyConcepts($\mathbb{A}$)	Retourne les concepts de l'ontologie	$\mathbb{C}$
getOntologyProperties($\mathbb{A}$)	Retourne les propriétés de l'ontologie	$\mathbb{P}$
getOntologyRelations($\mathbb{A}$)	Retourne les relations de l'ontologie	$\mathbb{P}_R$
getOntologyAttributes($\mathbb{A}$)	Retourne les attributs de l'ontologie	$\mathbb{P}_A$
getOntologyIndividuals($\mathbb{C}$,$\mathfrak{I}^C$)	Retourne les individus de l'ontologie	$\mathfrak{I}^C[\mathbb{C}]=\mathbb{I}$
getOntologyInstanceProperties ($\mathbb{P}$, $\mathfrak{I}^{\mathbb{P}}$)	Retourne les instances de propriétés de l'ontologie	$\mathfrak{I}^{\mathbb{P}}(\mathbb{P})$
getOntologyInstanceRelations ($\mathbb{P}$, $\mathfrak{I}^{\mathbb{P}}$)	Retourne les instances de relations de l'ontologie	$\mathfrak{I}^{\mathbb{P}}[\mathbb{P}_R]$
getOntologyInstanceAttributes ($\mathbb{P}$, $\mathfrak{I}^{\mathbb{P}}$)	Retourne les instances d'attributs de l'ontologie	$\mathfrak{I}^{\mathbb{P}}[\mathbb{P}_A]$
getSubConcepts(C,$\mathbb{A}$)	Retourne les sous concepts du concept C	$\subseteq \mathbb{C}$
getConceptProperties(C,$\mathbb{A}$)	Retourne les propriétés du concept C	$\subseteq \mathbb{P}$
getConceptRelations(C,$\mathbb{A}$)	Retourne les relations du concept C	$\subseteq \mathbb{P}_R$
getConceptAttributes(C,$\mathbb{A}$)	Retourne les attributs du concept C	$\subseteq \mathbb{P}_A$
getConceptIndividuals(C,$\mathfrak{I}^C$)	Retourne les individus du concept C	$\mathfrak{I}^C(C) \subseteq \mathbb{I}$
getPropertyInstances(P,$\mathfrak{I}^{\mathbb{P}}$)	Retourne les instances de la propriété P	$\mathfrak{I}^{\mathbb{P}}(P)$
getDomain(P,$\mathbb{A}$)	Retourne le domaine de la propriété donnée P	$\subseteq \mathbb{C}$
getRange(P,$\mathbb{A}$)	Retourne l'image de la propriété donnée P	$\subseteq \mathbb{C} \cup \mathbb{T}$
getDomainValue(i,P,$\mathfrak{I}^{\mathbb{P}}$)	Retourne la valeur de domaine de l'individu i pour la propriété P	$\in \mathbb{I}$
getRangeValue(i,P,$\mathfrak{I}^{\mathbb{P}}$)	Retourne la valeur d'image de l'individu i pour la propriété P	$\in \mathbb{I} \cup \mathbb{V}$
isDynamic(P_A,$\mathbb{A}$)	Retourne un booléen indiquant si l'attribut P_A est dynamique	*true/false*
getPredicateName(*prédicat*)	Retourne le nom du prédicat	$\in \mathbb{P}$
getDomainArgument(*prédicat*)	Retourne l'argument du domaine du prédicat (*Var* : ensemble de variables)	$\in \mathbb{I} \cup \mathbb{V} \cup Var$
getRangeArgument(*prédicat*)	Retourne l'argument de l'image du prédicat (*Var* : ensemble de variables)	$\in \mathbb{I} \cup \mathbb{V} \cup Var$
getRulesWithSameAntecedent (*Prédicats*)	Retourne les instances de *Règle-utilisateur* dont les valeurs de la propriété *Antécédent* sont identiques aux prédicats donnés	$\subseteq \mathfrak{I}^C[\mathbb{C}_{RU}]$
getRulesWithOppositeConsequent(*prédicat*)	Retourne les instances de *Règle-utilisateur* dont une valeur de la propriété *Conséquent* est opposée à celle du prédicat donné	$\subseteq \mathfrak{I}^C[\mathbb{C}_{RU}]$
getOppositeRules(i_{RU},$\mathfrak{I}^{\mathbb{P}}$)	Fait appel aux fonctions *getRulesWithSameAntecedent* et *getRulesWithOppositeConsequent* pour retourner les instances de *Règle-utilisateur* qui sont en contradiction avec i_{RU}	$\subseteq \mathfrak{I}^C[\mathbb{C}_{RU}]$

Tableau 1: Description des fonctions de manipulation de l'ontologie

3.1.4 *Développement de la termino-ontologie*

La richesse du langage naturel autorise l'usage de plusieurs dénominations pour faire référence à une même entité. Afin d'être en mesure de reconnaître une entité sous ses différentes formes linguistiques, il est nécessaire d'identifier et de regrouper les différentes dénominations de chaque instance de concept ou de propriété du domaine, ce qui revient à définir une terminologie du domaine.

Le recours à une terminologie permet de formaliser le sens des termes. De cette manière, chaque terme ou dénomination du texte peut être associé à une entrée dans la terminologie qui désigne le terme de référence.

Durant le peuplement d'ontologie à partir de textes, les termes à reconnaître sont ceux faisant référence aux instances de concepts ou de propriétés. La terminologie dans ce cas doit permettre la mise en relation des instances de concepts et propriétés avec leurs différentes formulations possibles dans les textes. On nomme une terminologie permettant ce type d'association : *termino-ontologie*.

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 une termino-propriété comme un terme désambiguïsé dont le sens permet de faire le lien entre une propriété de l'ontologie et les dénominations de ses instances dans les textes. Une termino-ontologie permet ainsi d'associer à une ontologie une composante linguistique afin d'établir un lien entre connaissances du texte et de l'ontologie. Afin d'établir ce lien, nous utilisons le formalisme S*imple Knowledge Organization System* (SKOS [5]) qui permet de définir une termino-ontologie et 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. Nous définissons les termino-concepts et termino-propriétés par deux propriétés prédéfinies *prefLabel* et *altLabel* et deux propriétés *implicit* et *partial* propres à notre approche. Ces quatre propriétés sont définies de la manière suivante :

- la propriété *prefLabel* définit la dénomination préférée d'un termino-concept ou d'une termino-propriété ;
- la propriété *altLabel* définit les dénominations alternatives d'un termino-concept ou d'une termino-propriété ;

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

– la propriété *implicit* n'est attribuée qu'aux termino-propriétés. Elle indique si la mention de l'instance de propriété associée peut être implicite dans les textes, et donc si un chemin *direct* peut être recherché entre les dénominations de ses domaine et image.
– la propriété *partial* n'est attribuée qu'aux termino-propriétés. Elle indique si la mention de l'instance de la propriété peut être impliquée dans un triplet partiel, et donc si un chemin *partiel* peut être recherché entre sa dénomination et l'une des dénominations de son domaine ou de son image.

3.1.5 *Analyse des textes : problèmes et solutions*

L'analyse de texte en langage naturel demande de faire face à deux problèmes majeurs : l'ambiguïté et l'implicite qui sont inhérents au langage naturel. Résoudre l'ambiguïté ou l'implicite nécessite de choisir ou de déduire le sens adéquat en fonction de connaissances extra-linguistiques. Pour cela le modèle de représentation doit faire plus que rendre compte de ce qui est spécifié dans les textes. Le modèle ontologique que nous proposons a pour objectif de fournir un cadre suffisamment formel pour guider l'interprétation des spécifications d'exigences. Nous montrons dans la suite du chapitre comment notre modélisation permet de résoudre ce type de problème, en explicitant les propriétés nécessaires et suffisantes à l'identification des individus auxquels il est fait référence dans les textes, et en associant les termes issus du texte à leur sens conceptuel à l'aide de la termino-ontologie, ainsi qu'en faisant usage des mécanismes de raisonnement autorisés par OWL.

3.1.6 *Conclusion*

Dans cette section, nous avons proposé un cadre générique pour la modélisation du comportement d'un système. L'objectif premier de ce cadre est de permettre la représentation des composants du système, de leurs caractéristiques et des règles de comportement. Cette modélisation repose sur une distinction claire entre deux types de concepts : *Composant* et *Type*. Cette distinction permet de rendre compte des différents niveaux de caractérisation des individus. Les individus du concept et sous-concepts de *Composant* correspondent à des référents individuels. Ils sont associés à des entités du monde réel. Les individus du concept et des sous-concepts de *Type* correspondent à des référents génériques ; ils ne définissent pas d'entités du monde mais les caractéristiques pouvant leur être associées. Les règles de comportement sont modélisées à l'aide d'instances du concept *Règle-utilisateur*. Ces

instances seront issues de la spécialisation des instances du concept *Patron-de-Règle* à partir de l'analyse des spécifications.

Le second objectif de notre modèle est de guider l'interprétation des spécifications. Cela suppose d'être en mesure de reconnaître d'une part les mentions faisant référence aux individus et d'autre part les mentions faisant référence à l'énonciation d'une règle utilisateur. Pour cela notre modèle définit pour chaque concept de l'ontologie les conditions nécessaires et suffisantes à l'identification de ces individus. L'association d'une termino-ontologie permet de lier le vocabulaire terminologique au vocabulaire conceptuel et ainsi d'associer une sémantique aux termes issus des spécifications. La reconnaissance des règles utilisateur est guidée par les instances du concept *Patron-de-Règle*.

Enfin, le dernier objectif de notre modèle ontologique est de permettre une traduction simple des spécifications d'exigences modélisées vers le langage de spécification formelle Maude (cf. section 4.6).

3.2 VÉRIFICATION DES EXIGENCES SOUS OWL

En section 1.8.3 nous avons défini trois types de vérifications à effectuer sur les règles de comportement. Dans cette section nous décrivons les vérifications possibles sous OWL ainsi que les mécanismes employés pour cela. Pour rappel, les trois propriétés que nous souhaitons vérifier sont : l'applicabilité de la règle ; la cohérence de la règle et la conformité des règles.

Nous proposons d'effectuer l'ensemble des vérifications qu'il est possible d'effectuer à l'aide et au sein de l'ontologie au cours de son instanciation. Pour cela, nous avons développé des fonctions écrites en *Java* en utilisant l'API Java *OWL-API*[6].

L'interface raisonneur *OWL-API* facilite l'utilisation de raisonneurs tels que FaCT++, HermiT, Racer et Pellet[7]. Seuls HermiT et Pellet supportent le raisonnement sur des règles SWRL et seul Pellet peut donner une justification pour toute inférence qu'il n'a pu effectuer. Nous utilisons donc Pellet comme raisonneur. Nous faisons aussi appel à *Jess*, un environnement de moteur de règles et de scripts permettant d'exécuter des requêtes *SQWRL*.

3.2.1 *Applicabilité de la règle*

Vérifier l'applicabilité de l'ensemble des règles revient à vérifier la complétude du modèle. En effet, une règle peut s'appliquer seulement

6. http ://owlapi.sourceforge.net/

7. Le lecteur peut se référer à [Dentler *et al.*, 2011] pour une comparaison des différents raisonneurs.

si l'ensemble des prédicats sur lesquels elle porte sont satisfaisables. Une règle étant formée d'une conjonction de prédicats, sa satisfaisabilité dépend de celle de chacun de ses prédicats. Un prédicat est satisfaisable s'il possède au moins une interprétation. Au sein de l'ontologie cela revient à vérifier si le prédicat en question peut être instancié, ou en d'autres termes, vérifier s'il existe au sein de l'ontologie au moins une instance de la classe sémantique décrite par le prédicat.

On distingue quatre types de prédicats à vérifier :

1. $P_k(v_1, v_2)$
2. $P_k(v_1, i_x)$
3. $P_k(i_x, v_1)$
4. $P_k(i_x, i_y)$

avec i_x et i_y des individus ou des littéraux (entiers, chaînes de caractères, ...), v_1 et v_2 deux variables.

Afin de vérifier l'applicabilité des règles de comportement modélisées dans l'ontologie, nous lançons avant chaque création d'une nouvelle règle, un ensemble de requêtes fondées sur l'utilisation de fonctions OWL-API et de requêtes SQWRL de la manière suivante :

Soient P_k un prédicat dont l'on souhaite vérifier la satisfaisabilité, $?v_x$ et $?v_y$ deux variables et i_x une constante (individu ou littéral). La partie gauche des règles SQWRL correspond aux prédicats à satisfaire. La partie droite sélectionne et renvoie l'ensemble des valeurs des arguments de la fonction *sqwrl :select* qui permettent de satisfaire la partie gauche de la règle.

1. $P_k(?v_x, ?v_y) \rightarrow$ *sqwrl :select*$(?v_x, ?v_y)$: renvoie l'ensemble (possiblement vide) des couples (v_x,v_y) associés à l'instance de propriété P_k ((v_x,v_y) $\in \mathfrak{I}^{\mathbb{P}}(P_k)$).
2. $P_k(i_x, ?v_y) \rightarrow$ *sqwrl :select*$(?v_y)$: renvoie l'ensemble (possiblement vide) des valeurs v_y associées à la valeur i_x au travers d'une instance de propriété P_k ((i_x,v_y) $\in \mathfrak{I}^{\mathbb{P}}(P_k)$).
3. $P_k(?v_y, i_x) \rightarrow$ *sqwrl :select*$(?v_y)$: renvoie l'ensemble (possiblement vide) des valeurs v_y associées à la valeur i_x au travers d'une instance de propriété P_k ((v_y,i_x) $\in \mathfrak{I}^{\mathbb{P}}(P_k)$).
4. La vérification du prédicat $P_k(i_x, i_y)$ quant à elle est effectuée par l'appel à la fonction *getDomainValue* (cf. tableau 1) suivant l'algorithme 1.

Lors de la définition du comportement d'un système, les interactions possibles entre composants du système sont en général déterminées par sa configuration technique. Celle-ci peut ne pas dépendre de l'utilisateur mais lui être imposée. Le résultat des requêtes ci-dessus

Input : i_x,i_y
$i_z \leftarrow$ getDomainValue(i_x,P,$\mathfrak{I}^{\mathbb{P}}$) ;
if $i_z = i_y$ **then**
retourne *vrai*;
end
Algorithme 1: Vérification de la satisfaction du prédicat $P_k(i_x, i_y)$

doit donc permettre de vérifier que les interactions décrites par l'utilisateur au sein des règles de comportement peuvent effectivement se produire. Dans le cas contraire l'utilisateur doit être averti afin de réviser ses spécifications d'exigences ou le cas échéant de proposer une modification de la configuration technique du système.

3.2.2 *Vérification de la cohérence*

Une fois l'applicabilité de la règle de comportement vérifiée, un raisonnement sur l'ontologie est effectué afin de vérifier que les instances participant à cette règle ne génèrent pas d'incohérences. Ce raisonnement est effectué à l'aide du raisonneur *Pellet* qui lorsqu'il déduit une incohérence, exhibe une justification. Cette justification correspond aux instances responsables. Il est donc possible via le lien entre instances et textes de retrouver les spécifications en cause, i.e. celles dont sont issues les instances responsables de l'incohérence.

Une fois la cohérence de l'ensemble des prédicats qui composent la règle vérifiée, la cohérence de la règle par rapport aux autres règles de l'ontologie doit être vérifiée. Une règle est dite cohérente par rapport aux autres règles si elle ne peut s'appliquer de manière simultanée avec une autre règle possédant une conséquence contradictoire. On peut distinguer deux cas de règles en contradiction :

1. deux règles ayant une même précondition ;
2. deux règles ayant des préconditions différentes mais qui peuvent s'appliquer simultanément.

Dans notre modélisation les règles de comportement sont représentées par des individus du concept *Règle-utilisateur*. Ces individus possèdent des valeurs pour les deux propriétés *Antécédent* et *Conséquent*. Ces valeurs correspondent aux instances de prédicats qui composent les règles de comportement identifiées à partir de l'analyse des spécifications. La vérification du premier cas d'incohérence est possible par l'appel à la fonction *getRulesWithOppositeConsequent* qui renvoie les règles de comportement possédant une conséquence opposée à celle de la nouvelle règle à vérifier. L'antécédent de chacune de ces règles est alors examiné afin de vérifier s'il est similaire à celui de la nou-

velle règle. Si aucune règle en contradiction n'est trouvée alors une nouvelle instance de *Règle-Utilisateur* est créée. Sinon les règles qui sont en contradiction sont exhibées. Comme chaque règle de comportement modélisée est liée à la spécification dont elle est issue, grâce à la propriété *Num-phrase*, l'utilisateur peut revenir aux spécifications afin d'apporter les corrections nécessaires à ses spécifications pour qu'elles soient cohérentes. L'ensemble du processus de vérification du cas (1) d'incohérence est résumé par l'algorithme 2.

Input : *new-rule*,$\mathfrak{I}^{\mathbb{P}}$
Output : *opposite-rules*
opposite-rules $\leftarrow \emptyset$;
C-prédicats $\leftarrow$ *getRangeValue(new-rule,Conséquent,*$\mathfrak{I}^{\mathbb{P}}$*)*;
for *each c-prédicat in C-prédicats* **do**
  $\mathcal{RU} \leftarrow$ *getRulesWithOppositeConsequent*(*c-prédicat*);
  for *each* $\mathfrak{ru}$ *in* $\mathcal{RU}$ **do**
    A-prédicats $\leftarrow$ *getRangeValue(new-rule,Antécédent,*$\mathfrak{I}^{\mathbb{P}}$*)*;
    opposite-rules $\leftarrow$ *opposite-rules* $\cup$
    getRulesWithSameAntecedent(*A-prédicats*);
    if *opposite-rules* $= \emptyset$ **then**
      createRuleInstance(new-rule,$\mathfrak{ru}$*)*;
    end
    else
      print("La règle *new-rule* est en contradiction avec l'ensemble de règles : " *opposite-rules*);
    end
  end
end

Algorithme 2: Fonction *checkOppositeRules*

La vérification du second cas d'incohérence, nécessite d'être en mesure d'identifier les antécédents différents pouvant être satisfaits de manière simultanée. Cette identification requiert l'exploration de l'ensemble des états du système. OWL ne permet pas ce type de vérification, les règles sont donc modélisées dans l'ontologie pour être : soit soumises ultérieurement à l'utilisateur qui pourra déterminer si les antécédents des règles en conflit potentiel peuvent s'appliquer de manière simultanée, soit traduites en spécifications d'un langage formel autorisant l'exploration des états d'un système (cf. chapitre 4).

3.2.3 *Vérification de la conformité des règles*

Les règles sont dites conformes si le comportement qu'elles décrivent permet de faire ce qui est attendu, et ne mène pas à des

états indésirables. La définition de ces états désirés ou non désirés ne dépend pas nécessairement des spécifications utilisateur. Le plus souvent, elle découle tout simplement du bon sens. Les cas désirés peuvent correspondre à des contraintes de sécurité, d'économie ou d'écologie. Un cas particulier d'état indésirable est l'état résultat de l'application simultanée de deux règles contradictoires. Vérifier la conformité des règles nécessite donc d'explorer les différents états auxquels conduisent les règles. OWL ne permettant pas ce type de vérification, elles seront effectuées par les outils du langage de spécifications formelles cibles (cf. chapitre 4).

Dans notre modélisation, les règles de comportement, pour représenter un comportement conforme, doivent porter sur des individus conceptualisant des objets réels correspondant à des référents individuels. Il est alors nécessaire de vérifier que les individus identifiés à partir des spécifications ont bien été liés à un objet du monde réel par la propriété *Num-identifiant*. Pour rappel, cette propriété identifie de façon unique un objet du monde (cf. section 3.1.1.2). Les individus possédant cette propriété sont ceux ayant été créés avant l'analyse des spécifications, ainsi que ceux issus de l'analyse des spécifications qui leur ont été associés par l'inférence d'une propriété *SameAs* (cf. section 2.1.4.2). Les individus de *Composant* ne possédant pas de valeur de propriété *Num-identifiant* sont alors ceux issus des spécifications qui n'ont pas pu être identifiés ou les individus de départ n'ayant pas été correctement créés.

Afin de renvoyer l'ensemble des individus de *Composant* non associés à un objet réel, nous exécutons les deux requêtes SQWRL (1) et (2), puis retournons l'ensemble de leurs différences (3) :

1. $\mathtt{Composant}(?C) \rightarrow \mathtt{sqwrl} : \mathtt{select}(?C)$
2. $\mathtt{Composant}(?C_{id})$ *Num-identifiant*$(?C_{id}, ?n) \rightarrow \mathtt{sqwrl} : \mathtt{select}(?C_{id})$
3. $C_{not-id} \leftarrow C \setminus C_{id}$

La requête (1) renvoie l'ensemble C ($C = \mathfrak{I}^{\mathbb{C}}$(*Composant*)) des individus du concept *Composant*, la requête (2) renvoie l'ensemble C_{id} ($C_id \subseteq \mathfrak{I}^{\mathbb{C}}$(*Composant*)) des individus du concept *Composant* identifiés comme associés à un objet réel du système (C_{id}). Le résultat de (3) est l'ensemble C_{not-id} ($\forall\, i \in C_{not-id} : i \in C \wedge i \notin C_{id}$) des individus de *Composant* n'ayant pas été identifiés. Ces individus, s'ils existent, doivent être retournés de manière à identifier la cause de leur non identification.

3.2.4 *Limite de la vérification sous OWL*

La formalisation des spécifications d'exigences a comme but final de vérifier la cohérence et la conformité des règles de comportement issues des spécifications d'exigences. Cette vérification doit s'opérer au niveau de chaque règle dans un premier temps, ce que permet OWL, puis porter sur les interactions possibles entre les règles, ce que OWL ne peut faire.

En effet, le raisonnement sous OWL est monotone, ce qui signifie que l'ajout de nouvelles connaissances ne fera qu'augmenter les déductions faites, les connaissances initiales ne pouvant être modifiées ou enlevées, ce qui empêche une représentation naturelle de l'évolution du changement d'états des individus. De plus, OWL ne permet pas de raisonner sur différentes séquences d'application de règles : lors du raisonnement, toutes les règles sont appliquées en même temps. Vérifier l'interaction des règles demande d'être en mesure d'explorer leurs différentes possibilités d'application. La vérification de l'interaction des règles ne peut donc être effectuée sous OWL. Une transition vers un langage formel permettant l'exploration de modèle est donc nécessaire.

Nous avons choisi Maude [Clavel *et al.*, 2002], un langage de spécifications formelles fondé sur les logiques de réécriture. Les règles de comportement sont donc représentées dans l'ontologie afin d'être traduites en spécifications Maude, suite au peuplement de l'ontologie et de la vérification du modèle ontologique. Les vérifications permises par Maude et le processus de traduction de notre modèle ontologique en une spécification Maude sont décrites dans le chapitre 4.

3.2.5 *Conclusion*

Dans cette section nous avons présenté les différentes vérifications possibles sous OWL. Grâce à son formalisme logique et ses différents mécanismes de raisonnement, OWL permet de vérifier un bon nombre de propriétés qui nous intéressent. OWL semble s'avérer être à la hauteur de nos attentes en fournissant un cadre suffisamment formel pour modéliser et vérifier la cohérence des différents prédicats d'une règle de comportement. Néanmoins, le raisonnement monotone de OWL limite la portée de ses vérifications. Il n'est par exemple, pas possible de représenter le changement dynamique de l'état d'un individu. La vérification de l'évolution de l'état d'un système est donc en dehors de la portée d'OWL. Le modèle résultant du peuplement de l'ontologie est cohérent et suffisamment formel pour en proposer une traduction automatique. Nous envisageons donc une traduction vers le langage de spécifications formelles *Maude*. Les enjeux d'une tra-

duction de OWL à Maude sont décrits dans le chapitre 4. La section suivante décrit notre approche de peuplement d'ontologie.

3.3 PEUPLEMENT DE L'ONTOLOGIE

3.3.1 *Introduction*

Dans cette section, nous décrivons notre approche de peuplement d'ontologie et détaillons comment nos choix de modélisation (cf . section 3.1.1) permettent tout au long du processus de peuplement, d'exploiter au mieux les mécanismes de raisonnement pour reconnaître de manière précise les mentions d'instances de propriétés dans les spécifications, de classer et d'identifier les individus au sein de l'ontologie et de faire apparaître la présence d'erreurs de spécifications au plus tôt.

Dans notre approche, l'ontologie impliquée dans le processus de peuplement définit les contraintes liées au domaine. Elle définit en particulier les propriétés *définissantes* décrivant les conditions d'identification des individus (cf section 1.8.4). Il en découle que les individus de l'ontologie sont identifiables à partir de leurs valeurs de propriétés *définissantes*. Identifier les instances de propriété à partir des textes permet alors d'identifier de manière fiable les individus qui y participent. Nous centrons donc notre approche d'identification d'instances sur l'identification d'instances de propriétés dans les spécifications. L'identification des instances de concepts, ou individus, est effectuée par raisonnement au sein de l'ontologie. Cette approche a l'avantage de mettre à profit les connaissances sémantiques définies par l'ontologie pour identifier de manière précise et univoque les individus participant au comportement du système à réaliser.

L'objectif du peuplement est de faire le lien entre les mentions d'instances dans les textes et leur conceptualisation dans l'ontologie. Dans les textes, les instances de concepts et de propriétés sont dénotées par des termes. Identifier la présence de mentions d'instances de propriétés dans les textes revient alors à identifier les termes qui les dénotent. De manière générale, une instance de propriété correspond à un triplet (instance de propriété, instance du domaine, instance de l'image) [Makki *et al.*, 2009; Takhirov *et al.*, 2012].

Dans la suite du document, nous suivons les notations ci-dessous pour représenter les termes pouvant dénoter des instances de l'ontologie.

- nous notons t_C un terme dénotant une instance de concept i_C ;
- nous notons t_P un terme dénotant une instance de propriété i_P ;
- nous notons (t_P, t_{C_D}, t_{C_R}) le triplet de termes dénotant l'instance de propriété $i_P(i_{C_D}, i_{C_R})$ avec i_{C_D} et i_{C_R} deux instances du domaine D et de l'image R de la propriété P.

Identifier une instance de propriété revient à identifier le triplet de termes qui la dénote. Ce triplet est de la forme (t_P, t_D, t_R) avec t_P, t_D

et t_R trois termes qui dénotent respectivement les mentions d'une instance de propriété modélisée dans l'ontologie, d'une instance du domaine et d'une instance de l'image de la propriété en question. La méthode d'extraction d'instances que nous proposons est donc guidée par l'identification de triplets de termes dans les textes.

Nous distinguons deux types de triplets que nous nommons : complets et partiels.

DEFINITION 17: Un **triplet complet** contient trois termes qui font référence à une instance de propriété ainsi qu'aux instances de son domaine et de son image. Il est de la forme (t_P, t_D, t_R). Par exemple, à partir de la phrase *Si le capteur de la cuisine détecte un mouvement, allumer la lumière* on peut extraire le triplet (*détecte*, *capteur*, *mouvement*) référant à une instance de propriété (*Détecte*) pour laquelle les référents de ses instances de domaine (*Capteur*) et d'image (*Phènomène*) sont explicitement mentionnés dans le texte.

DEFINITION 18: Un **triplet partiel** a pour vocation de reconnaître des propriétés pour lesquelles une des deux instances de domaine ou d'image n'est pas explicitement spécifiée dans le texte. Il est de la forme $(t_P, ?t, t_R)$ ou $(t_P, t_D, ?t)$ avec ?t une référence inconnue, ou implicite. Les triplets partiels visent à permettre d'identifier des mentions d'instances de concepts implicites dont l'identité sera inférée par un raisonnement ontologique. Par exemple, à partir de la phrase *Lorsqu'un mouvement est détecté dans la cuisine, allumer la lumière*, on peut extraire le triplet (*détecte*,?t,*mouvement*) référant à une instance de propriété *Détecte* pour laquelle le référent de son instance de domaine *Capteur* n'est pas mentionné dans le texte contrairement au référent de son instance d'image *Phènomène*.

Nous soutenons que guider le peuplement de l'ontologie par l'identification d'instances de propriétés permet de résoudre des cas d'ambiguïté de termes et d'informations implicites pour lesquels les instances de concepts associées sont déduites par inférence au sein de l'ontologie à partir des définitions des *propriétés de classification et d'identification* d'individus. De la sorte, la classification et l'identification d'instances de concepts ne reposent pas seulement sur la reconnaissance de leurs mentions dans les textes mais aussi sur les propriétés sémantiques auxquelles elles sont associées.

L'approche que nous proposons a comme objectif d'être suffisamment générique pour s'adapter aisément à différents domaines d'application, dès lors que l'on peut les décrire par une ontologie ainsi qu'à d'autres langues dès lors qu'elles possèdent un analyseur syntaxique et qu'une termino-ontologie amorce soit fournie. Nous montrons que cette approche permet d'identifier de manière fiable des

instances de propriétés, et dans un deuxième temps d'inférer les instances de concepts qui leurs sont associées.

3.3.2 *Ontologie de départ*

L'ontologie peut contenir des instances dont l'existence est préalable au processus de peuplement (cf section 2.3.3). Dans notre modèle, les composants et leurs types conditionnent le comportement du système et ne dépendent pas des spécifications utilisateur qui ne font que décrire les règles de comportement des différents composants. Aussi, les instances correspondant aux individus du concept *Composant* et du concept *Type* doivent être créées de façon préalable à l'analyse des spécifications. Le processus de peuplement consiste alors à lier les mentions faisant référence aux composants et leurs types aux référents conceptuels 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 issus des spécifications aux individus préexistants dans l'ontologie lorsqu'ils sont identiques [8], i.e. qu'ils représentent une même entité. Ces liaisons seront permises par l'application d'inférences fondées sur l'exploitation des propriétés d'identification définies par notre modèle ontologique.

3.3.3 *Règles d'extraction d'instances de propriétés*

Les triplets correspondant aux instances de propriétés que l'on recherche dans les textes sont extraits à l'aide de règles d'extraction d'instances de propriété acquises automatiquement par amorçage à partir de l'ontologie de départ, d'une termino-ontologie SKOS « amorce » et d'un corpus d'apprentissage (cf. figure 14). Ces règles correspondent à des formes lexico-syntaxiques récurrentes augmentées de connaissances sémantiques (ontologiques).

Le processus d'amorçage consiste à effectuer une première itération à partir d'un petit ensemble de termes *amorces*, de l'ontologie OWL et d'un corpus d'apprentissage. Cette première itération permet d'acquérir des règles d'extraction de termes (cf. section 3.3.5). L'application de ces règles sur le corpus d'apprentissage permet l'acquisition de nouveaux termes. À chaque itération la termino-ontologie s'étend, i.e. son nombre de termes augmente, puis la termino-ontologie étendue sert à l'acquisition de nouvelles règles d'extraction de termes. Les itérations se succèdent ainsi, jusqu'à ce que plus aucun nouveau terme

8. Deux individus sont identiques si leurs valeurs de propriétés d'identification sont identiques

ou aucune nouvelle règle ne soit acquis. Alors, l'acquisition de règles d'extraction d'instances de propriété peut commencer.

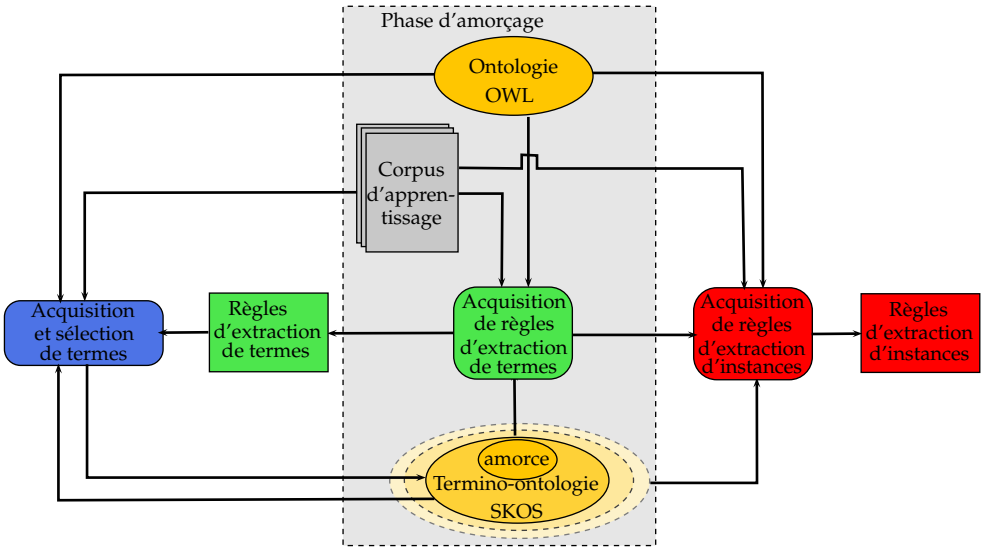


FIGURE 14: Acquisition de règles d'extraction par amorçage

Les règles d'extraction d'instances de propriétés ont comme objectif de reconnaître dans les textes les mentions de propriétés conceptualisées dans l'ontologie. Ces mentions sont représentées par des triplets de termes. Chaque règle doit alors être en mesure d'identifier un triplet de termes. Les triplets sont formés de trois termes dénotant la propriété ainsi que ses domaine et image. Ces informations sont formalisées au sein de l'ontologie. Nous distinguons deux types de règles d'extraction :

DEFINITION 19: Une **règle d'extraction de triplets complets** doit extraire des triplets complets de la forme (t_P, t_D, t_R), afin de reconnaître des instances de propriétés potentielles de la forme $i_P(i_D, i_R)$ avec i_P un nom de propriété dénoté par t_P, i_D un individu dénoté par t_D et i_R un individu ou un littéral dénoté par t_R.

Chaque règle d'extraction de triplets complets a pour objectif de représenter un des contextes possibles de formulation d'une instance de propriété. Dans notre approche, ce contexte est syntaxico-sémantique. Il est d'abord syntaxique, car il exploite les caractéristiques d'un arbre de dépendances syntaxiques, il est aussi sémantique car il est défini entre les trois ensembles termino-ontologiques T_P, T_D, T_R qui représentent respectivement, les termes qui dénotent les instances de la propriété P, de son domaine D et de son image R.

DEFINITION 20: Une **règle d'extraction de triplets partiels** doit extraire des triplets partiels. Chaque triplet partiel est de la forme $(t_P, ?t, t_D)$ ou $(t_P, t_D, ?t)$. Il contient deux termes ainsi qu'un inconnu $?t$. Ces règles doivent capturer le contexte syntaxico-sémantique qui

lie les termes dénotant une instance de propriété avec les termes dénotant soit une instance de son domaine soit de son image. Les instances de propriétés candidates sont alors de la forme $i_P(i_D, ?i_R)$ ou $i_P(?i_D, i_R)$.

Dans le cas des règles d'extraction de triplets partiels, l'instance inconnue doit être identifiée au sein de l'ontologie. Le contexte défini par cette seconde catégorie de règles est de fait moins contraint, i.e. moins précis, mais néanmoins nécessaire au traitement de l'implicite inhérent aux textes en langage naturel. Par exemple, dans la phrase : *Switch on my computer when I get in the house*[9], l'utilisateur fait référence à deux entités de façon implicite. L'entité qui permet de détecter la présence d'une personne (un capteur) ainsi qu'une entité permettant d'allumer un ordinateur (un actionneur). Dans notre approche de conceptualisation, ces deux entités représentent des composants du système. Tout l'intérêt de la définition d'un cadre formel de raisonnement est alors de permettre la résolution de problèmes d'identification d'entités dans les textes lorsque celles-ci ne sont pas explicitement spécifiées. Un exemple de triplets partiels qui devront être extraits de la précédente phrase sont : *(get,* $?t_S$*, house)*

et *(Switch on,*$?t_A$*,computer)* avec $?t_S$ et $?t_A$ les mentions implicites d'une instance du concept *Sensor* et d'une instance du concept *Actuator* respectivement.

Certaines propriétés P peuvent être dénotées simplement à partir du contexte syntaxico-sémantique entre les termes de leurs ensembles de domaine T_D et d'image T_R (cf. section 2.3.6.4), sans que le contexte n'inclue leurs lexicalisations. Ce type de propriété doit alors être défini sur des ensembles de domaine et d'image suffisamment spécialisés pour ne laisser une fois les paires de termes qui les dénotent reconnues, aucune ambiguïté sur le type de la propriété. Cela équivaut à spécialiser de manière systématique les propriétés de l'ontologie. À titre d'exemple, une propriété générique *Type* peut se décliner en plusieurs propriétés plus spécifiques ; définies sur des classes sémantiques plus restreintes, telles que la propriété *Perceived-type* ayant pour domaine le concept *Sensor* et pour image le concept *Phenomenon-Type* ou la propriété *Managed-type* ayant pour domaine le concept *Actuator* et pour image le concept *Phenomenon-Type*. L'idée est alors de restreindre suffisamment les domaine et image de chaque propriété de l'ontologie de sorte que les paires de termes qui les dénotent limitent au possible les ambiguïtés.

Reconnaître les propriétés dont les mentions peuvent être implicites demande l'application des deux types de règles d'extraction de

9. Spécification issue du corpus d'évaluation, acquis via notre plate-forme d'acquisition de spécifications d'exigences dans le domaine des environnements intelligents http ://perso.limsi.fr/sadoun/Application/fr/SmartHome.php

triplets complets et de triplets partiels, afin d'identifier aussi bien des formulations comportant une lexicalisation de la propriété « *the sensor is of the kind*[10] *movement* » que des formulations ou la mention à la propriété est implicite « *A movement sensor is fixed in the entrance* ».

3.3.4 *Acquisition de règles d'extraction d'instances de propriétés*

You shall know a word by the company it keeps [Firth, 1957]

Firth [Firth, 1957] affirme que l'on peut connaître un mot à partir de la compagnie qu'il entretient , cette compagnie qu'il évoque représente le contexte dans lequel évoluent les termes du texte. Nos règles d'extraction ont pour vocation de capturer le contexte dans lequel apparaissent les termes dénotant des instances de propriété. L'idée est de faire émerger des textes les régularités syntaxiques et sémantiques traduisant les mentions d'instances de propriétés conceptualisées dans l'ontologie. Nous fondons l'acquisition des règles d'extraction sur l'exploitation de dépendances syntaxiques, issues de l'analyse syntaxique de corpus d'apprentissage et des connaissances ontologiques modélisées. Le contexte est alors déterminé non seulement par les termes, mais aussi par les dépendances syntaxiques, qui les lient et les classes sémantiques qu'ils dénotent.

Dans la littérature, la définition du contexte de formulation d'une instance de propriété suit le plus souvent le postulat selon lequel les chemins syntaxiques qui ont tendance à se connecter aux mêmes paires d'ensembles de termes véhiculent possiblement le même sens (cf. section 2.3.6.4). Dans notre approche, nous proposons d'identifier le contexte syntaxico-sémantique d'une instance de propriété à partir d'ensembles de triplets qui la dénotent. Cette notion de triplet de termes a l'avantage de représenter de manière plus fine les mentions de propriétés dans les textes que celles de paires de termes typiquement utilisées pour l'acquisition d'instances de propriétés. En outre, l'inclusion de termes dénotant la propriété dans le chemin permet une analyse plus précise et par ailleurs, l'identification des instances de propriétés contenant de l'implicite.

Plus exactement, nous nous intéressons aux chemins syntaxiques que composent ces dépendances lorsqu'elles connectent les triplets de termes. Les règles sont alors générées à partir des caractéristiques des chemins syntaxiques les plus fréquents et de l'exploitation des connaissances sémantiques modélisées par l'ontologie.

Notre méthode vise à acquérir les règles d'extraction de manière automatique. Pour cela nous faisons appel à l'ontologie, à une termino-ontologie SKOS comme amorce et un à corpus d'apprentissage. Le

10. *kind* est définie dans notre termino-ontologie SKOS comme une dénomination de la propriété *Perceived-type*.

processus d'acquisition des règles d'extraction est décrit par l'algorithme 3. L'analyse syntaxique du corpus d'apprentissage donne lieu à la création d'un arbre de dépendances syntaxiques pour chacune de ses phrases (cf. section 3.3.4.1). Puis, l'acquisition commence par l'obtention de l'ensemble des termes dénotant chaque propriété de l'ontologie à partir de la termino-ontologie SKOS, de même que les deux ensembles de termes dénotant ses domaine et image (cf. section 3.3.4.2). L'ensemble des chemins syntaxiques entre les termes des trois ensembles de termes sont alors extraits (cf. section 3.3.4.3) et les chemins les plus fréquents sont sélectionnés (cf. section 3.3.4.4). Enfin, une règle d'extraction est générée à partir de chaque chemin sélectionné (cf. section 3.3.4.5).

3.3.4.1 *Création d'arbres de dépendances syntaxiques*

Chaque phrase du corpus est analysée syntaxiquement à l'aide du Stanford Parser [11]. L'analyse syntaxique permet d'engendrer pour chacune des phrases, son arbre de dépendances syntaxiques (cf. figure 15). Un arbre de dépendances est un arbre représentant les fonctions syntaxiques entre les mots d'une phrase. Les nœuds d'un arbre de dépendances sont les mots de la phrase. Chaque nœud est étiqueté par la catégorie morpho-syntaxique du terme qu'il contient. Les branches sont les dépendances syntaxiques. Les nœuds sont reliés deux à deux par les dépendances syntaxiques qui constituent des liens orientés.

DEFINITION 21: Une **dépendance syntaxique** lie deux nœuds. Le premier est nommé *directeur* et le second *dépendant*. La figure 15 illustre l'arbre de dépendances syntaxiques de la phrase *"When a person moves into the kitchen, switch on the light."* [12]. Un exemple de dépendance syntaxique issue de cette arbre est `pobj(into, kitchen)` dont les éléments sont définis ci-dessous :

- `pobj` : nom de la dépendance ;
- `into` : mot contenu dans le nœud directeur ;
- `kitchen` : mot contenu dans le nœud dépendant.

En domaine de spécialité, ce sont les termes et non les mots qui sont considérés comme les unités de sens atomiques. Dans notre approche, il est donc important de prétraiter l'arbre de dépendances afin que ses nœuds ne contiennent plus des mots mais des termes. L'étape de prétraitement n'est pas décrite ici. Elle sera détaillée en section 5.5.

À partir de l'arbre syntaxique, nous construisons un ensemble de toutes les dépendances syntaxiques contenues dans la phrase. Les

11. http ://nlp.stanford.edu/software/lex-parser.shtml.
12. produit à l'aide de DependenSee.jar http ://chaoticity.com/dependensee-a-dependency-parse-visualisation-tool/

Input : corpus, ontologie, skos
Output : $R_{Complet}$,$R_{Partiel}$
$Paths \leftarrow \emptyset$;$halfPaths \leftarrow \emptyset$;$R_{Complet} \leftarrow \emptyset$,$R_{Partiel} \leftarrow \emptyset$;
//Analyse syntaxique du corpus
DependencyTrees $\leftarrow$ *StanfordParser*(corpus);
//Parcours de l'ensemble des arbres de dépendances
for *each depsTree in DependencyTrees* **do**
  $P \leftarrow getProperties(ontologie)$;
  for *each* p *in* P **do**
    // Retourne les termes associés à la propriété p
    $T_{prop} \leftarrow getTerms(p, skos)$;
    // Retourne les concepts associés au domaine de p
    $C_D \leftarrow getDomain(p, ontologie)$;
    // Retourne les concept associés à l'image p
    $C_R \leftarrow getRange(p, ontologie)$;
    for *each* c_D *in* C_D **do**
      // Retourne les termes associés au concept c_D
      $T_{Domain} \leftarrow getTerms(c_D, skos)$;
      for *each* c_R *in* C_R **do**
        // Retourne les termes associés au concept c_R
        $T_{Image} \leftarrow getTerms(c_R, skos)$;
        // Chemins entre triplets complets
        $Paths \cup$ extractPaths(depsTree, T_{Prop}, T_{Domain}, T_{Image});
        // Chemins entre triplets partiels
        // - - - Domaine implicite - - -
        $halfPaths \cup$ checkHalfPaths(depsTree, T_{Prop}, T_{Image});
        // - - - Image implicite - - -
        $halfPaths \cup$ checkHalfPaths(depsTree, T_{Prop}, T_{Domain});
$MFPaths \leftarrow getMostFrequentPaths(Paths)$;
for *each* $path$ *in* $MFPaths$ **do**
  $R_{Complet} \leftarrow R_{Complet} \cup createCorrespondingRule(path)$;
$MFHalfPaths \leftarrow getMostFrequentHalfPaths(halfPaths)$;
for *each* $halfpath$ *in* $MFHalfPaths$ **do**
  $R_{Partiel} \leftarrow R_{Partiel} \cup createCorrespondingRule(halfpath)$;

Algorithme 3: Acquisition des règles d'extraction

termes contenus dans ces dépendances sont mis sous leur forme lemmatisée.

3.3.4.2 *Obtention des ensembles termino-ontologiques*

L'objectif est de rechercher des chemins entre des triplets de termes qui dénotent les mêmes ensembles termino-ontologiques. À cette étape, il s'agit donc d'obtenir pour chaque propriété, l'ensemble des termes qui la dénotent, ainsi que les ensembles de termes qui dénotent les instances de son domaine et les instances de son image. Pour cela, nous faisons appel à la fonction *getTerms* qui étant donné le nom d'une classe sémantique, renvoie l'ensemble des termes qui lui sont associés dans la termino-ontologie SKOS.

3.3.4.3 *Extraction des chemins syntaxiques*

DEFINITION 22: Un **chemin syntaxique** est composé d'une succession de dépendances syntaxiques liant des termes [13] dans un arbre de dépendances. Il est à noter que le nombre de dépendances peut être égal à 1.

On dit qu'une dépendance syntaxique dep_k constitue un chemin entre t_x et t_y si elle lie t_x et t_y. On distingue deux cas possibles :

1. $dep_k(t_x,t_y)$
2. $dep_k(t_y,t_x)$

On dit que deux dépendances syntaxiques dep_i et dep_j forment un chemin syntaxique entre t_x et t_y si l'une d'elle contient t_x et l'autre t_y et qu'elles partagent un nœud commun [14]. Formellement, si dep_i et dep_j sont deux dépendances syntaxiques, alors on dira que dep_i-dep_j est un chemin syntaxique si et seulement si l'un des cas suivant est vrai :

1. $dep_i(t_x, \mathbf{a})$ et $dep_i(\mathbf{a}, t_y)$
2. $dep_i(t_x, \mathbf{a})$ et $dep_j(t_y, \mathbf{a})$
3. $dep_i(\mathbf{a}, t_x)$ et $dep_j(\mathbf{a}, t_y)$
4. $dep_i(\mathbf{a}, t_x)$ et $dep_j(t_y, \mathbf{a})$

Dans notre approche, nous construisons les chemins syntaxiques à l'aide d'un algorithme récursif (cf. algorithme 5) qui exploite les deux définitions ci-dessus. Il est à noter que l'ordre des dépendances syntaxiques n'est pas important dans les chemins syntaxiques car elles constituent des liens orientés.

13. Issus de la phase de prétraitement. Ils sont composés d'un ou de plusieurs mots

14. Ce nœud commun ne correspond évidemment ni à celui de t_x ni à celui de t_y.

Dans la figure 15, il est possible de distinguer un chemin syntaxique entre le terme *moves* et le terme *kitchen*. Ce chemin est en gras sur la figure 15 et est le suivant $\texttt{prep}(\texttt{moves}, \texttt{into}) - \texttt{pobj}(\texttt{into}, \texttt{kitchen})$.

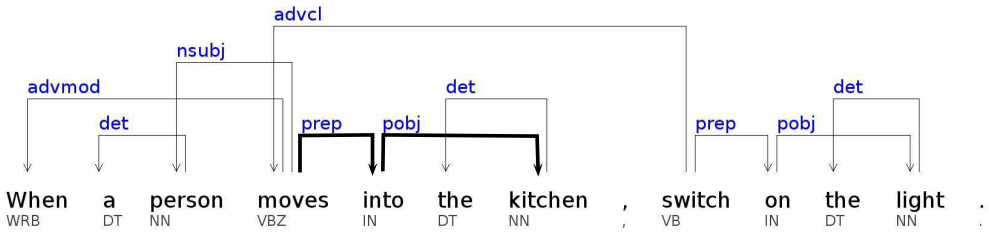


FIGURE 15: Arbre de dépendances syntaxiques

EXTRACTION DE CHEMINS SYNTAXIQUES Deux types de règles sont à acquérir. Pour cela deux fonctions ont été définies : *extractPaths* et *checkHalfPaths*. *extractPaths* recherche des chemins entre trois ensembles de termes dénotant les instances d'une propriété, de son domaine et de son image. Elle prend comme argument l'ensemble des dépendances syntaxiques ainsi que les trois ensembles de termes. Elle est de la forme *extractPaths(DepsTree,* T_{Prop}*,* T_{Domain}*,* T_{Image}*)* (cf. algorithme 4). Les chemins contenant un triplet complet (t_P, t_D, t_R) sont constitués de deux chemins entre triplets partiels (t_D, t_P) et (t_P, t_R).

Afin de construire un chemin entre triplet, la fonction *extractPaths* fait deux appels successifs à la fonction *checkHalfPath* (cf. algorithme 4) qui vérifie l'existence d'un chemin entre une paire de termes donnée.

La fonction *checkHalfPath(DepsTree,* t_1*,* t_2*, Paths)* parcourt pour chaque paire de termes (t_1, t_2) l'ensemble des dépendances syntaxiques *DepsTree* à la recherche de chemins potentiels et construit chaque chemin de manière récursive (cf. algorithme 5). *checkHalfPaths* recherche des chemins entre deux ensembles de termes dénotant les instances d'une propriété et de son domaine, d'une propriété et de son image ou encore du domaine et de l'image de la propriété, si la mention de celle-ci peut être implicite. Elle prend comme argument l'ensemble des dépendances syntaxiques ainsi que les deux ensembles de termes à lier. Elle est de la forme *checkHalfPaths(DepsTree,* T_{Prop}*,* T_{Image}*)*, *checkHalfPaths(DepsTree,* T_{Prop}*,* T_{Domain}*)* ou encore *checkHalfPaths(DepsTree,* T_{Domain}*,* T_{Image}*)* (cf. algorithme 4).

Lors de la création des chemins syntaxiques, de manière similaire à Takhirov *et al.* [2012] qui masquent les noms d'entités nommées d'un patron d'extraction pour le rendre générique, chaque terme des dépendances est remplacé par sa catégorie syntaxique et s'il fait partie

du triplet de termes recherchés, on y associe aussi les classes sémantiques (concepts ou propriétés) qu'il dénote. De cette façon, les chemins extraits ne contiennent plus que des éléments génériques. Les chemins épurés de la sorte sont nommés *chemins génériques*. Dans une phrase il peut exister plus d'un chemin entre une paire de termes. Dans ce cas, chaque chemin est extrait. Seuls sont conservés les chemins partiels qui ne sont pas contenus dans un chemin complet d'un même arbre de dépendances.

Input : DepsTree, T_P, T_D, T_I, skos
Output : Paths
Paths $\leftarrow \emptyset$;
enable-direct-link $\leftarrow$ *false*;
// Vérifie si la mention de la propriété peut être implicite
if *enableDirectLinks(*T_P,skos*)* **then**
  enable-direct-link $\leftarrow$ *true*;
end
for *each* t_p *in* T_P **do**
  for *each* t_D *in* T_D **do**
    for *each* t_I *in* T_I **do**
      // Vérifie si un chemin direct entre t_D et t_R peut dénoter T_P
      if *enable-direct-link* **then**
        // Recherche un chemin *direct* ne contenant pas t_p
        checkHalfPath(*DepsTree*, t_D, t_R, *Paths*);
      end
      checkHalfPath(*DepsTree*, t_D, t_P, *Paths*);
      checkHalfPath(*DepsTree*, t_P, t_I, *Paths*);
    end
  end
end

Algorithme 4: Fonction extractPaths

3.3.4.4 *Sélection des chemins les plus fréquents*

Une fois l'ensemble des chemins génériques extraits, il reste à sélectionner les plus pertinents. L'hypothèse première est que plus un chemin est fréquent, plus il a de chance de refléter une propriété sémantique. À l'inverse les chemins très peu fréquents risquent plus d'être incorrects. Une fois toutes les phrases du corpus analysées, les chemins génériques sont regroupés en fonction du triplet (propriétés de l'ontologie) ayant guidé leur extraction. Au sein de chaque groupe, les chemins sont comparés en fonction des caractéristiques de leurs dépendances syntaxiques. Deux chemins sont considérés identiques,

Input : *DepsTree*, t_1, t_2, *Paths*
Output : *Paths*
$path \leftarrow \emptyset$;
if $\exists\ dep(t_1, t_2) \in DepsTree$ **then**
$path \leftarrow path \cup dep(t_1, t_2)$
else
if $\exists\ dep(t_1, a) \in DepsTree$ **then**
if $checkHalfPaths(DepsTree, a, t_2, path) \neq \emptyset$ **then**
$path \leftarrow path \cup dep(t_1, a)$
if $checkHalfPaths(DepsTree, t_2, a, path) \neq \emptyset$ **then**
$path \leftarrow path \cup dep(t_1, a)$
else
if $\exists\ dep(a, t_2) \in DepsTree$ **then**
if $checkHalfPaths(DepsTree, t_1, a, path) \neq \emptyset$ **then**
$path \leftarrow path \cup dep(a, t_2)$
if $checkHalfPaths(DepsTree, a, t_1, path) \neq \emptyset$ **then**
$path \leftarrow path \cup dep(a, t_2)$
$Paths \leftarrow Paths \cup path$;

Algorithme 5: Fonction checkHalfPaths

s'ils contiennent les mêmes dépendances syntaxiques et que ces dépendances portent sur les mêmes catégories syntaxiques et classes sémantiques. Formellement, deux chemins syntaxiques Ch_1 et Ch_2 sont identiques si :

$$\forall dep(c_1, c_2) \in Ch_1 \rightarrow dep(c_1, c_2) \in Ch_2 \wedge$$
$$\forall dep(c_1, c_2) \in Ch_2 \rightarrow dep(c_1, c_2) \in Ch_1$$

avec c_1 et c_2 deux nœuds formés d'une catégorie syntaxique et éventuellement d'une classe sémantique. Les groupes de chemins identiques sont dénombrés. Deux critères de sélection sont alors utilisés : la fréquence d'un chemin et sa longueur. Pour chaque propriété sémantique, les n (paramètre à fixer) chemins identiques les plus fréquents sont retournés. Plus un chemin entre deux termes sera long et moins il sera probable que ces deux termes soient liés sémantiquement. Aussi, seuls les chemins dont la longueur est inférieure à l (paramètre à fixer) sont retenus. Enfin, les règles d'extraction sont générées à partir des caractéristiques des chemins retournés.

3.3.4.5 *Génération des règles d'extraction*

La génération des règles d'extraction se fait de manière automatique. Afin de capturer le contexte syntaxico-sémantique reflétant la mention d'une propriété conceptuelle, les règles sont fondées sur l'exploitation des caractéristiques des chemins génériques les plus fré-

quents. La génération d'une règle d'extraction exploite les trois catégories d'information suivantes :

- dépendances syntaxiques (Dep) ;
- catégories morpho-syntaxiques (Cat) ;
- classe sémantique ou ensemble termino-ontologique (TO).

La forme générale d'un chemin générique pour un triplet (t_P, t_D, t_R) est la suivante :

$$Dep_1(Cat_D\text{-}TO_D, Cat_1) \wedge \ ... \ \wedge \ Dep_i(Cat_j, Cat_P\text{-}TO_P) \wedge Dep_{i+1}(Cat_P\text{-}TO_P, Cat_{j+1}) \wedge ... \wedge Dep_k(Cat_k, Cat_R\text{-}TO_R)$$

avec Cat_D-TO_D la combinaison catégorie syntaxique et classe sémantique du terme du domaine, Cat_P-TO_P la combinaison catégorie syntaxique et classe sémantique du terme de la propriété et Cat_R-TO_R la combinaison catégorie syntaxique et classe sémantique du terme de l'image.

Les règles d'extraction générées à partir des chemins syntaxiques sélectionnés sont de la forme :

- dépendances syntaxiques -
$Dep_1(t_D, t_1) \wedge .. \wedge Dep_j(t_P, t_j) \wedge .. \wedge Dep_k(t_k, t_R) \wedge$

- catégories morpho-syntaxiques -
$Cat_1(t_D) \wedge .. \wedge Cat_P(t_P) \wedge .. \wedge Cat_j(t_R) \wedge$

- ensembles termino-ontologiques -
$TO_I(t_D) \wedge .. TO_P(t_p) \wedge .. \wedge TO_R(t_R)$

- instance de propriété candidate -
$\rightarrow TO_P(t_D, t_R)$

Le chemin syntaxique $prep(move^{15}, into) \wedge pobj(into, kitchen)$ illustré en figure 15 a comme chemin générique *prep(VB-Event,IN-Detected-in)* $\wedge$ *pobj(IN-Detected-in,NN-Location)*. Il est constitué de :

- dépendances syntaxiques -
$prep(move, into) \wedge pobj(into, kitchen)$

- catégories morpho-syntaxiques -
Verb$(move) \wedge$ *Prep*$(in) \wedge$ *Noun*$(kitchen)$

15. *move* est la forme lemmatisée de *moves*.

- ensembles termino-ontologiques -
TO_{Event} [16](move) $TO_{Detected-in}$ [17](in) $TO_{Location}$ [18](kitchen)

Les dépendances syntaxiques sont transformées en prédicats, les catégories morpho-syntaxiques et termino-ontologiques sont transformées en contraintes. Cela permet d'engendrer la règle d'extraction de la propriété $Detected-in(Phenomenon, Location)$ avec $TO_{Location}$, $TO_{Detected-in}$ et $TO_{Phenomenon}$ trois ensembles de termes issus de l'ontologie lexicale :

$$\begin{aligned} &prep(t_D, t_P) \wedge pobj(t_P, t_R) \wedge \\ &isVerb(t_D) \wedge isPrep(t_P) \wedge isNoun(t_R) \wedge \\ &TO_{Event}(t_D) \wedge TO_{Detected-in}(t_P) \wedge TO_{Location}(t_R) \\ &\rightarrow Detected-in(t_D, t_R) \end{aligned}$$

L'application des règles d'extraction a deux objectifs. Tout d'abord, identifier dans les spécifications, les instances de propriété candidates. L'identification des instances est très dépendante de la termino-ontologie acquise au préalable. Appliquées sur un corpus, les règles d'extraction permettent d'étendre la termino-ontologie.

3.3.5 *Extension de la termino-ontologie*

Extraction des termes

La quantité et la pertinence des termes utilisés sont déterminantes pour la performance d'un système d'extraction d'information. Dans le cas d'une extraction guidée par l'ontologie, la termino-ontologie est d'autant plus importante que c'est elle qui fait le lien entre connaissances linguistiques et ontologiques. L'acquisition de la terminologie se fait par amorçage, c'est-à-dire que l'on construit la termino-ontologie de façon itérative à partir d'un petit ensemble de termes de départ. À chaque itération la terminologie courante est utilisée pour l'acquisition de règles d'extraction (cf. section 3.1.4). Puis les règles extraites sont appliquées sur un corpus d'apprentissage afin d'extraire de nouveaux termes. Ces règles servent alors à l'expansion des ensembles terminologiques représentant les classes sémantiques de l'ontologie. Chaque règle d'extraction capture le contexte dans lequel apparaissent trois ensembles sémantiques. Elle peut donc engendrer trois applications, avec comme objectif d'extraire à chaque appli-

16. termes dénotant une instance du concept *Event*
17. termes dénotant une instance de la propriété *Detected-in*
18. termes dénotant une instance du concept *Location*.

cation les termes dénotant un des trois ensembles en exploitant les deux autres.

DEFINITION 23: Une **règle d'extraction de termes** R peut être vue comme une fonction prenant comme arguments deux ensembles sémantiques et ayant comme résultat l'ensemble sémantique recherché. On définit une règle d'extraction de la manière suivante :

Soit R et $T_{Domaine}$, T_{Image}, T_{Prop} trois ensembles de termes dénotant des classes sémantiques conceptualisées dans l'ontologie.

1. $R : T_{Domaine} \times T_{Prop} \rightarrow T_{Image}$
2. $R : T_{Image} \times T_{Prop} \rightarrow T_{Domaine}$
3. $R : T_{Domaine} \times T_{Image} \rightarrow T_{Prop}$

Dans le cas de règles d'extraction portant sur des triplets partiels les fonctions d'extraction R ne prennent qu'un seul ensemble sémantique. Chacune n'engendre alors que deux applications :

1. $R : T_{Domaine} \rightarrow T_{Prop}$
2. $R : T_{Prop} \rightarrow T_{Domaine}$

ou

1. $R : T_{Image} \rightarrow T_{Prop}$
2. $R : T_{Prop} \rightarrow T_{Image}$

Sélection des termes

Chaque règle d'extraction représente un des contextes syntaxico-sémantiques d'énonciation d'une instance de propriété. Chacune de ses applications permet l'extraction d'un ensemble de termes (cf. figure 16 (1)) représentant l'un des trois ensembles terminologiques participant à l'instance, i.e. dénotant une instance de la propriété, de son domaine ou de son image. Plusieurs règles d'extraction peuvent être nécessaires pour représenter l'ensemble des contextes syntaxico-sémantiques d'énonciation d'une instance de propriété. Ces règles partagent alors les mêmes trios d'ensembles termino-ontologiques (cf. figure 16 (2)). De même, d'autres types de règles représentant d'autres instances de propriétés peuvent permettre d'extraire des ensembles termino-ontologiques similaires (cf. figure 16 (3)). Les termes d'une même classe sémantique peuvent donc être extraits par différentes règles, intuitivement il semble raisonnable de regrouper les termes extraits des textes par les classes sémantiques auxquelles ils font référence.

La pertinence des termes extraits est en général estimée en fonction de leur fréquence (cf. figure 17 a).

Notre intuition est que les termes les plus pertinents peuvent être extraits par plusieurs règles d'extraction du même type, i.e. représentant le contexte syntaxico-sémantique d'une même instance de propriété, alors que les termes les moins pertinents ne seront extraits que

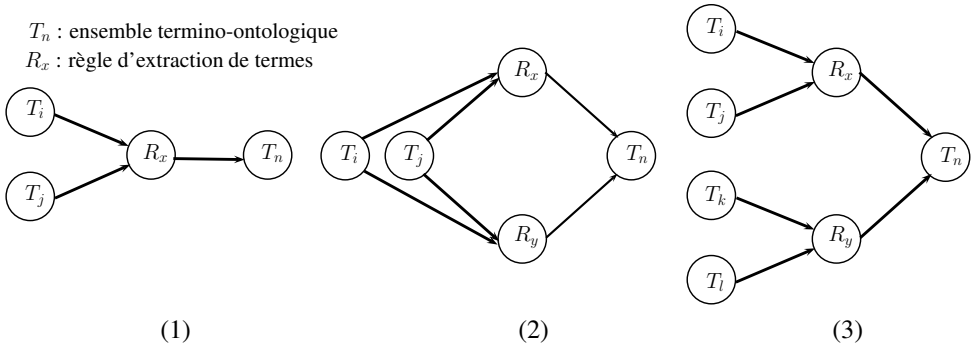


FIGURE 16: Applications des règles d'extraction de termes

par quelques unes. C'est-à-dire qu'un terme qui apparaît dans différents contextes syntaxico-sémantiques dénotant une même instance, est plus susceptible d'être pertinent qu'un terme qui n'apparaît que dans un seul contexte. Cette intuition est renforcée par l'hypothèse empirique que différentes règles d'extraction devraient commettre des erreurs indépendantes [Carlson *et al.*, 2010].

Notre méthode de sélection des termes pertinents est fondée sur la distinction des différents contextes dans lesquelles les termes apparaissent. Pour une même classe, chaque règle est appliquée séparément. Les termes les plus fréquents de chaque ensemble (issus d'une même règle) sont reclassés en fonction du nombre de contextes auxquels ils sont associés (cf. figure 17 b).

La pertinence des termes est calculée par la combinaison de leur fréquence et du nombre de contextes syntaxico-sémantiques dénotant une même classe sémantique dans lesquels ils apparaissent.

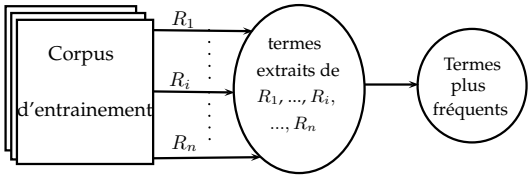
La pertinence d'un terme t extrait par des règles R_i est calculée selon la formule suivante :

$$\mathtt{Pertinence}(t) = (\sum_{i=1}^{n} \mathtt{freq}(t, R_i) * w_{R_i}) * (n!)$$

avec $\mathtt{freq}(t, R_i)$ la fréquence du terme t extrait par la règle R_i, w_{R_i} le poids de la règle qui sera plus important si la règle est issue d'un triplet complet plutôt qu'un triplet partiel et n le nombre de règles d'extraction d'une même classe sémantique à partir desquelles le terme a été extrait. Cette formule a pour objectif de favoriser les termes extraits par plusieurs règles. Ainsi, un terme extrait deux fois par trois règles du même type a un score plus élevé qu'un terme extrait six fois par la même règle.

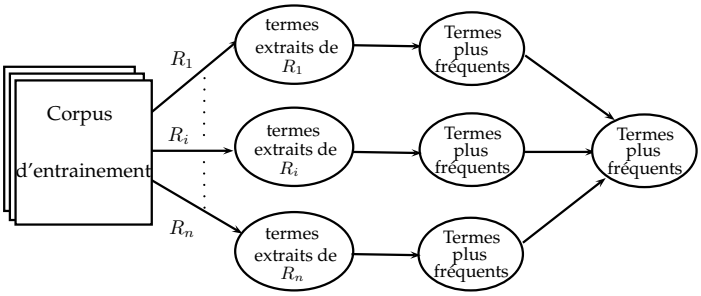
3.3.6 *Processus de peuplement*

L'objectif du peuplement est la représentation des règles de comportement reconnues dans les spécifications d'exigences. La reconnais-



R_i: Règles d'extraction d'un même ensemble sémantique.

a) Méthode fondée sur la fréquence des termes



b) Méthode tenant compte des différents contextes syntaxico-sémantique

FIGURE 17: Extraction de nouveaux termes

sance de ces règles est guidée par les patrons de règles représentés dans l'ontologie par les individus du concept *Patron-de-Règle*. L'objectif de l'analyse des spécifications est de reconnaître les prédicats (instances de propriétés) définis par ces patrons, puis de les instancier à partir des instances reconnues dans les spécifications. L'ensemble du processus de peuplement est décrit par l'algorithme 6 et suit les étapes détaillées ci-après :

3.3.6.1 *Application des règles d'extraction*

Chacune des phrases des spécifications est analysée et son arbre de dépendances généré. L'ensemble des dépendances syntaxiques est construit à partir de l'arbre de dépendances. L'ensemble des règles d'extraction d'instances sont appliquées de manière successive sur l'ensemble de dépendances afin d'en extraire les instances de propriété candidates.

3.3.6.2 *Sélection des instances*

La sélection des instances candidates est guidée par les individus du concept *Patron-de-Règle*. L'idée est de rechercher parmi les instances extraites, celles correspondant aux prédicats pour chaque patron de règles. Pour chaque instance de *Patron-de-Règle*, l'ensemble des prédicats spécifiques (contenant des variables ou représentant une super-propriété cf. section 3.1.1.4) est parcouru puis chaque prédicat

Input : spécifications, ontologie, skos
for *each sentence* S *in spécifications* **do**
// Retourne l'arbre de dépendances;
Tree ← getDependencyTree(S)
// Retourne les dépendances syntaxiques
Deps ← getDependencies(Tree) ;
//Extrait les instances de propriétés candidates
$I_{P_{Candidates}}$ ← ApplyExtractionRules(Deps);
//Retourne les individus de *Patron-de-Règle*
$\mathbb{I}_{C_{PR}}$ ← *getConceptIndividuals*(*Patron-de-Règle*,$\mathcal{I}^{C}$);
for *each* i_{PR} *in* $\mathbb{I}_{CPR}$ **do**
$I_{P_{Found}} \leftarrow \emptyset$; $I_{P_{Missing}} \leftarrow \emptyset$;
//Retourne les prédicats spécifiques des patrons de règles
PR_{Dyn} ← getDynamicPart(i_{PR});
for *each* pr_{Dyn} *in* PR_{Dyn} **do**
//Vérifie si le prédicat correspond à une instance candidate
if *MatchIn(*pr_{Dyn}, $I_{P_{Candidates}}$*)* **then**
$I_{P_{Missing}} \leftarrow I_{P_{Missing}} \cup pr_{Dyn}$;
else
for *each* $i_{p_{Candidates}}$ *in* $I_{P_{Candidates}}$ **do**
if $i_{p_{Candidates}}.name = pr_{Dyn}.name$ **then**
//Instancie les variables du prédicat
$i_{p_{new}} \leftarrow instanciatePredicate(pr_{Dyn}, i_{p_{Candidates}})$;
$I_{P_{Found}} \leftarrow I_{P_{Found}} \cup i_{p_{new}}$;
if $I_{P_{Found}} \neq \emptyset$ **then**
//Crée les instances de propriété reconnues
createInstances($I_{P_{Found}}$);
//Vérifie la cohérence de l'ontologie
if isConsistent(ontologie) **then**
//Vérifie qu'aucun prédicat spécifique de i_{PR} ne manque
if $I_{P_{Missing}} = \emptyset$ **then**
//Associe les valeurs d'instances aux prédicats
$i_{UR} \leftarrow mapInstanceAndPredicate(i_{PR}, I_{P_{Found}})$;
//Vérifie qu'aucune règle contradictoire n'ait été trouvée
opposite-rules← *getOppositeRules*(i_{UR});
if *opposite-rules*= $\emptyset$ **then**
//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 !");
else
print(Les instances $I_{P_{Found}}$+ "génèrent des incohérences.");

Algorithme 6: Peuplement de l'ontologie

est comparé aux instances de propriétés candidates. Cette comparaison a pour objectif de vérifier si suffisamment de prédicats spécifiques ont été reconnus dans la phrase pour faire apparaître la présence d'une règle utilisateur, spécialisant un des patrons de règles. Lorsqu'une correspondance est établie entre un prédicat spécifiques et une instance de propriété candidate, le prédicat est instancié à partir des informations de l'instance candidate correspondante. L'instanciation consiste à remplacer certains arguments spécifiques par la valeur correspondante dans l'instance de propriété ou définir l'instance de la propriété (qui spécialise la super-propriété) à créer dans la partie conséquent (cf. figure 13).

3.3.6.3 *Création des instances de propriétés*

Les instances de propriétés candidates retenues à l'étape précédente sont créées dans l'ontologie. Une étape préalable à la création des instances de propriétés est le nommage des instances participant à la propriété. Quatre cas de figure se présentent :

1. Les instances de types simples tels que les entiers ou les chaînes de caractères gardent la valeur reconnue dans le texte ;
2. Les individus dénotant une instance du concept *Type*, c'est-à-dire un référent générique sont nommés à partir de leur formulation préférée, définie dans la termino-ontologie SKOS ;
3. Les individus dénotant une instance du concept *Composant*, c'est-à-dire un référent individuel, ne peuvent être reconnus directement à partir de leurs mentions dans les textes (cf. section 2.1.4.2), leur identité ne pouvant être déterminée qu'à partir de leur valeur de propriété. Ils sont nommés de façon générique à partir du numéro de phrase et de nœud syntaxique dans lesquels ils apparaissent. Les noms de ces individus sont de la forme : *num phrase-num nœud*. Par exemple, si le terme dénotant l'individu est issu de la phrase numéro 11 et du nœud syntaxique numéro 4 alors l'individu est nommé *11-4*. De cette façon chaque individu est nommé de manière unique, ce qui permet de se défaire de l'ambiguïté inhérente aux termes.
4. Les individus correspondant aux mentions implicites dans les triplets partiels sont nommés de façon générique à partir des numéros de phrase et de nœud syntaxique dans lesquels apparaît le terme qui dénote l'instance de propriété à laquelle ils sont associés. Par exemple, si l'individu est issu de la mention implicite $?t_D$ du triplet $(t_P,?t_D,t_R)$ alors il est nommé à partir des numéros de phrase et de nœud du terme t_P qui dénote une instance de propriété.

Ainsi, l'exemple de triplet partiel *(Switch on,*$?t_A$*,computer)* issu de la spécification *Switch on my computer when I get in the house* (cf. section 3.3.3), donne lieu à la création de l'instance de propriété *Turn-on*(*24-1,24-3*) où *24-1* correspond à l'instance du domaine de la propriété *Turn-on* [19] et *24-3* correspond à l'instance de l'image de la propriété *Turn-on*. L'instance *24-1* est nommée à partir du numéro de la phrase (24) et du numéro de nœud du terme *Switch on* (3) car sa mention dans la spécification est implicite ($?t_A$). L'instance *24-3* est nommée à partir du numéro de la phrase et du numéro de nœud du terme *computer*.

Nommer les individus issus de mentions explicites et implicites de manière générique, i.e. à partir des numéros de phrase et de nœuds syntaxiques permet d'une part, de s'abstraire de la forme linguistique des mentions explicites qui dénotent les individus dans les textes et d'autre part, d'éliminer la part d'implicite des formulations langagières. En effet, au sein de l'ontologie chaque individu se définit par ses valeurs de propriétés et non par sa formulation dans le texte, ce qui autorise un traitement équivalent des individus issus de mentions explicites ou implicites. Le raisonnement sur l'ensemble des individus permet ensuite de les classer et de les identifier de manière unique.

3.3.6.4 *Classification et identification des individus*

À ce stade, les classes sémantiques des individus impliqués dans les nouvelles instances de propriétés ne sont pas connues ; par défaut ils appartiennent tous à la classe *Thing* [20]. L'application d'un raisonnement sur l'ensemble de l'ontologie a alors pour objectifs : a) de reconnaître la classe sémantique de chaque nouvel individu ; b) d'identifier pour chaque individu, les individus de l'ontologie représentant la même entité ; c) vérifier la cohérence des nouvelles connaissances ajoutées et inférées (cf. section 3.3.6.5).

La classification des individus s'effectue dans l'ontologie par l'application d'un raisonnement sur leurs valeurs de propriétés. Ce raisonnement tient compte des conditions de classification (cf. section 2.1.4.1). Durant le processus de peuplement un individu est classé essentiellement de deux manières [21] : en fonction du domaine ou de l'image des propriétés auxquels il est associé ou à partir des conditions nécessaires et suffisantes, définies pour chaque concept. Afin d'illustrer la première manière, nous reprenons l'exemple d'instance de propriété *Turn-on*(*24-1,24-3*) créée à l'étape précédente. À partir des contraintes

19. Le terme *Switch on* dénote la propriété *Turn-on*.
20. Dans OWL *Thing* correspond au concept parent de tous les concepts.
21. La troisième manière consiste à l'associer aux concepts parents des concepts auxquels il est déjà associé.

de domaine et d'image de la propriété *Turn-on* les deux instances *24-1* et *24-3* sont associées respectivement aux concepts *Actuator* et *Physical-process*.

L'identification des individus se fait par raisonnement sur les conditions d'identification (cf. section 2.1.4.2), le raisonnement porte alors sur les propriétés fonctionnelles ou inverses fonctionnelles pour les individus pouvant être distingués par une seule propriété ou à l'aide de règle SWRL lorsque les individus nécessitent plus d'une propriété pour être distingués des autres. Des exemples de cas d'identification d'individu sont donnés au chapitre 5.

Suite au processus de classification chaque individu peut se voir associer un ou plusieurs concepts.

3.3.6.5 *Vérification de la cohérence de l'ontologie instanciée*

Chaque individu se voit attribuer lors de sa création trois valeurs de propriétés. Ces valeurs correspondent à son numéro de phrase pour la propriété *Num-phrase*, à son numéro de nœud pour la propriété *Num-node* et à sa formulation dans les spécifications *Linguistic-ref*. Ces connaissances permettent de faire le lien entre instances conceptuelles et références linguistiques. L'objectif est de pouvoir si nécessaire retrouver aisément les mentions d'instances dans les textes.

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 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*. Le lien conceptuel entre les instances et leurs mentions dans le texte permet alors de pointer les erreurs de spécifications à corriger. Dans le cas où les instances créées n'introduisent pas d'incohérences, la création de règles de comportement est envisageable.

3.3.6.6 *Création des règles de comportement*

Les règles de comportement spécifiées par un utilisateur sont représentées par les instances du concept *Règle-utilisateur*. Rappelons que chaque instance du concept *Règle-Utilisateur* est une spécialisation d'une instance du concept *Patron-de-Règle* qui représente un type de règle générique. La création de règles de comportement exploite les instances de propriétés extraites des spécifications. À cette étape,

la cohérence de ces instances a été vérifiée et leur correspondance avec les prédicats du patron de règle courant établie. Avant la création d'une nouvelle instance de règle, deux vérifications restent à faire. Tout d'abord il faut vérifier que l'ensemble des prédicats spécifiques du patron de règle courant a été reconnu et que l'instance de règle résultat n'est en contradiction avec aucune instance de règles de l'ontologie (cf. section 3.1.1.4) algorithme 2). Si ces deux dernières conditions sont vérifiées, alors une instance du concept *Règle-utilisateur* est créée. À cette instance sont associées deux valeurs de propriétés. Le numéro de phrase dont elle est issue via la propriété *Num-phrase*, ainsi que le patron de règle associé via la propriété *Patron-Règle*.

3.3.7 *Conclusion*

Dans cette section, nous avons décrit une approche de peuplement d'ontologie qui permet de représenter la sémantique des exigences de manière formelle à partir de spécifications en langage naturel. Notre approche est centrée sur l'identification de mentions d'instances de propriété dans les textes. Les mentions d'instances sont reconnues comme des triplets de termes dans les textes. Ces triplets sont identifiés par des règles d'extraction, acquises automatiquement et fondées sur les chemins syntaxiques récurrents reliant les termes qui désignent les instances de concept et de propriété. Ces règles exploitent des connaissances lexicales, syntaxiques et sémantiques. Elles identifient les instances de propriété, même si leur domaine ou image est implicite. En outre, ces règles permettent de lever l'ambiguïté des termes extraits en capturant le contexte sémantique dans lequel ils apparaissent. En guidant l'identification des instances de concepts par l'identification d'instances de propriété, il est possible de les reconnaître dans leur contexte. Ainsi, leur identification ne s'appuie pas uniquement sur une reconnaissance lexicale mais prend aussi en compte les rôles sémantiques des individus. De plus, nous montrons comment utiliser le pouvoir d'inférence OWL, qui permet de classer les instances de concepts et de les identifier d'une manière unique. L'approche proposée a été conçue pour être indépendante du domaine et facilement adaptable à d'autres langues. Elle requiert en entrée une ontologie formelle, un corpus d'apprentissage ainsi qu'une termino-ontologie amorce. Les résultats obtenus lors du processus de peuplement automatique de notre modèle ontologique sont donnés et discutés dans le chapitre 5.

Dans ce chapitre, nous avons proposé un modèle ontologique permettant de décrire le comportement d'un système de manière précise et formelle, nous avons décrit les vérifications autorisées par ce modèle ainsi que les possibilités que celui-ci pouvait offrir lors d'un

processus de peuplement automatique à partir de spécification d'exigences en langage naturel. Le passage des spécifications d'exigences au modèle ontologique constitue une première étape dans le processus de production de spécifications formelles. La seconde étape correspond au passage de notre modèle ontologique à une spécification formelle Maude. Cette dernière est détaillée dans le chapitre suivant.

4

DU MODÈLE ONTOLOGIQUE AU LANGAGE DE SPÉCIFICATIONS FORMELLES

Sommaire

« C'est avec l'intuition que nous trouvons et avec la logique que nous prouvons. »

Inspirée d'une citation de Henri Poincaré.

4.1 INTRODUCTION

L'ontologie OWL développée lors de l'étape précédente décrit les composants d'un système, leurs caractéristiques ainsi que les règles régissant leur comportement. OWL permet la vérification de la plupart des propriétés qui nous intéressent. Néanmoins le raisonnement monotone auquel est soumis OWL ne permet pas la modification des connaissances existantes de l'ontologie, ce qui limite la possibilité de représenter de manière naturelle l'évolution du changement d'état des entités modélisées. La validation des spécifications d'exigences nécessite la simulation du comportement du système qu'elles décrivent. Le but est de vérifier que tous les états désirés peuvent être atteints et que l'ensemble des états non désirés ne l'est pas. Une des méthodes de vérification privilégiée dans ce cas est le *model-checking*, méthode de vérification automatique, exhaustive et systématique de tous les états possibles d'un système à partir d'un état initial. En cas de détection d'une erreur, un contre-exemple est généré pour illustrer de quelle façon la propriété à vérifier a été violée. Lorsque ce type de vérification est inclus dans le processus de conception d'un système, il permet la détection précoce d'erreurs et de comportements indésirables.

Notre objectif est donc de choisir un langage de spécifications formel cible qui autorise l'application de techniques de model-checking, ayant des propriétés et une sémantique proches de celles des ontologies, de sorte à permettre une transformation directe. Le langage formel recherché doit être fondé sur un formalisme logique permettant l'expression de connaissances représentées en logique de description. La représentation des concepts et individus dans OWL est, syntaxiquement et sémantiquement, très similaire à la description des classes d'objets et instances en programmation orienté objet [Koide *et al.*, 2005] [1]. Le langage formel cible doit alors être proche du paradigme orienté objet.

Notre choix s'est orienté vers le langage Maude car celui-ci remplit ces exigences. Maude [Clavel *et al.*, 2002, 2011] est un langage de spécification et de programmation fondé sur la logique de réécriture [Meseguer, 1990] qui intègre une panoplie d'outils de validation

1. Pour une comparaison des deux paradigmes, le lecteur peut se référer au travail de thèse décrit dans [Koide, 2010].

et de vérification [Clavel *et al.*, 2007] dont un modèle checker [Eker *et al.*, 2004]. Martí-Oliet et Meseguer [2002] ont montré que Maude représente un très bon cadre logique et sémantique dans lequel différentes logiques et différents formalismes peuvent être exprimés et exécutés. De plus, il est particulièrement bien adapté pour la spécification de systèmes concurrents orientés objet [Romero *et al.*, 2007] et permet de décrire l'état d'un système et son comportement de façon mathématique et rigoureuse. En outre, divers travaux [Romero *et al.*, 2007; Rivera *et al.*, 2009; Rusu, 2013] ont montré que Maude offre un moyen simple, naturel et précis de spécifier les modèles et les méta-modèles [2]. Au fil des années, Maude s'est avéré être un bon environnement pour le prototypage rapide [Meseguer, 2010] et l'aide au développement et à la validation de modèles UML/OCL [Clavel et Egea, 2006b], de façon plus générale, de méta-modèles MOF [3] [Boronat et Meseguer, 2010]. De nombreux travaux ont également exploré son utilisation pour la spécification de modèles décrits par des langages du web sémantique [Verdejo *et al.*, 2005; Senanayake *et al.*, 2005; Elenius *et al.*, 2008; Huang *et al.*, 2009; Song *et al.*, 2012; Gueffaz, 2012] tels que RDF ou OWL.

Ce chapitre a pour objet de décrire comment le modèle ontologique proposé au chapitre précédent autorise une transformation simple vers un langage de spécifications formelles tel que Maude. Nous commençons par une description du problème et des enjeux d'une telle transformation et justifions le choix de Maude comme langage de spécifications formelles cible, puis nous discutons certaines approches ayant traité le passage entre un langage de modélisation et un langage de spécifications formelles. Nous donnons ensuite une brève description de Maude axée sur ses aspects orienté objet, puis nous décrivons les éléments de OWL qui devront être transformés, ainsi que leur correspondance avec les éléments de Maude. Enfin, nous détaillons une approche de transformation du modèle ontologique en un modèle Maude puis du modèle Maude aux spécifications formelles Maude correspondantes. Cette transformation doit permettre de vérifier et valider le comportement concurrent des règles issues des spécifications d'exigences.

2. Un méta-modèle décrit de façon formelle la structure communes à l'ensemble des modèles d'un même domaine : concepts, objets, relations, attributs etc. et se décrit lui-même

3. Le Meta-Object Facility (MOF) est un standard de l'Object Management Group qui permet la représentation des méta-modèles et leur manipulation. UML est défini par un méta-modèle MOF

4.2 DES LANGAGES DE MODÉLISATION AUX LANGAGES DE SPÉCIFICATIONS FORMELLES

Notre objectif est la transformation de l'ontologie OWL du comportement système obtenue à partir de l'analyse des spécifications d'exigences en langage naturel en spécifications formelles Maude. La littérature portant sur le passage de OWL à Maude n'étant pas très étendue, nous généralisons la problématique au passage d'un langage de modélisation à un langage de spécifications formelles. Nous nous intéressons d'abord aux approches proposant une transformation entre un langage de modélisation, qui de même que OWL, est proche du paradigme orienté objet vers le langage Maude, puis aux approches qui proposent une traduction de OWL vers un langage de spécifications formelles.

Parmi les langages de modélisation les plus connus et utilisés, nous pouvons citer UML (et ses dérivés SysML, MSC, LSC), RDF, RDFS et OWL. Le langage UML s'appuie sur une approche orientée objet, et propose un ensemble de diagrammes permettant de modéliser la structure et le comportement d'un système. La notation UML est largement fondée sur l'utilisation de diagrammes. Néanmoins, ces diagrammes ne fournissent pas le niveau de concision et d'expressivité qu'un langage textuel formel peut offrir [Clavel et Egea, 2006b]. Aussi les diagrammes UML sont souvent complétés par la définition de contraintes OCL[4] qui ne peuvent être exprimées graphiquement, et que le modèle doit satisfaire. Le fait qu'UML ne possède pas une sémantique précise est un sérieux inconvénient [Chama *et al.*, 2013] qui a poussé un grand nombre de chercheurs à s'intéresser à la transformation d'un modèle UML en un langage de spécifications formelles tel que B[5] [NGUYEN, 1998; Ledang et Souquières, 2001; Laleau et Mammar, 2001; Truong, 2006], Alloy [Anastasakis *et al.*, 2007], CSP[6], Object-Z[7] ou les réseaux de Petri [Gulan *et al.*, 2013] ou encore Maude [Clavel et Egea, 2006b; Mokhati et Badri, 2009; Roldan et Duran, 2010; Duràn et Roldàn, 2012].

La majorité des travaux qui s'attaquent à un passage de UML à Maude traitent systématiquement la transformation du diagramme de classes car c'est lui qui décrit les composants du système. Ils se distinguent ensuite par la traduction d'autres diagrammes, de

4. Le langage d'expression de contraintes OCL permet de spécifier des invariants, des pré- et post-conditions à vérifier [Roldan et Duran, 2010].

5. B est une méthode formelle de développement logiciel qui couvre le processus logiciel à partir de la spécification abstraite jusqu'à l'application exécutable.

6. CSP, est un langage de description de modèles d'interaction. Il correspond à une algèbre de processus permettant de modéliser l'interaction de systèmes.

7. http ://itee.uq.edu.au/~smith/objectz.html

contraintes OCL ou par les correspondances choisies entre éléments UML et Maude. En règle générale, dans une traduction de UML à Maude, chaque type simple UML (entier, réel, booléen, string) est associé à la sorte (type) Maude prédéfinie correspondante [Clavel et Egea, 2006b; Roldan et Duran, 2010; Duràn et Roldàn, 2012]. Une première approche consiste alors, à l'image de celle proposée par Clavel et Egea [2006b] en une traduction du modèle UML et des contraintes OCL [Clavel et Egea, 2006a] en une spécification équationnelle. Clavel et Egea [2006b] représentent chaque classe UML par une sorte, chaque objet UML par une constante de sorte, chaque attribut d'une classe par un opérateur et proposent de créer pour chaque association entre paires de classes, deux opérateurs. Les contraintes OCL sont représentées par des termes booléens. La validation des diagrammes de classes et d'objets[8] par rapport aux contraintes consiste alors à vérifier si les termes booléens se réécrivent en vrai ou faux. L'inconvénient de cette approche est qu'elle ne préserve pas l'aspect orienté objet que nous souhaitons exploiter dans notre approche de traduction. Dans [Mokhati et Badri, 2009] la traduction proposée mêle équations et règles de réécriture. Elle consiste à produire une spécification formelle Maude à partir de l'étude de différents diagrammes UML (de classes, de cas d'utilisation, de communication et d'états-transitions). La traduction produit un module orienté objet Maude pour la description de chaque classe ainsi qu'un module orienté objet pour la description de l'ensemble des messages possibles entre deux objets. Le reste des spécifications donne lieu à la génération de modules standards. Le plus souvent, les diagrammes de classes sont traduits sous forme de modules orientés objet Maude, et l'état du système par des configurations d'objets [Romero *et al.*, 2007; Rivera *et al.*, 2009; Boronat et Meseguer, 2010] ou par des configurations d'objets et de messages [Roldan et Duran, 2010; Duràn et Roldàn, 2012; Chama *et al.*, 2013]. Dans ce cas, les classes objets et messages UML sont associées aux classes, objets et messages du module orienté objet Maude. Quant aux attributs de classes UML, ils sont représentés par des opérateurs Maude et dans la définition des classes Maude. Quelques différences résident dans le paramétrage des opérateurs en fonction de l'objectif du système à construire. Duràn et Roldàn [2012] s'intéressent à la validation de contraintes OCL incluses dans les diagrammes de classes et en proposent une traduction UML/OCL vers Maude afin de simuler des modèles UML en exécutant leurs spécifications Maude correspondantes. Cette traduction a pour objectif de permettre la vérification des scénarios qui répondent ou qui violent des contraintes

8. Un diagramme d'objets représente les objets (instances de classes) utilisés dans le système.

données [Durán *et al.*, 2011]. Pour cela, ils représentent les contraintes OCL (pré- et post-condition et invariant) à l'aide d'opérateurs Maude qu'ils définissent à l'aide d'équations. L'utilisation d'équations limite la représentation de l'évolution d'état des objets et donc du système.

La plupart des approches de traduction des langages d'ontologie RDF et OWL vers des langages de spécifications formelles exploitent leurs aspects orientés objet. Par exemple [Verdejo *et al.*, 2005; Senanayake *et al.*, 2005; Huang *et al.*, 2009] proposent en général une traduction vers un module orienté objet Maude. Au contraire Elenius *et al.* [2008] décrivent les individus, les concepts, et les propriétés OWL à l'aide d'opérateurs Maude. Ils représentent les individus par des constantes, les concepts par des opérateurs qui prennent en argument un individu et qui renvoient un booléen. les ObjectProperty par des opérateurs entre deux individus qui renvoient un booléen et les DataProperty par des opérateurs entre un individu et un type simple qui renvoient un booléen. Cette dernière approche a l'inconvénient de ne pas préserver la notion d'individu, ou d'entité dont les interactions et les changements d'états déterminent le comportement du système.

Senanayake *et al.* [2005] investiguent la traduction de contraintes représentées dans une ontologie OWL par des règles SWRL vers des règles de réécriture Maude. Ils considèrent les individus OWL définis dans l'ontologie comme équivalent à l'état initial d'un processus de réécriture. L'ensemble des individus OWL est transformé en une liste d'objets dont les attributs correspondent à leurs propriétés dans l'ontologie. Puis, cet ensemble d'objets est regroupé à l'aide de la sorte *Configuration* qui correspond à un multi-ensemble d'objets et de messages, de sorte à préserver la sémantique orienté objet de OWL. Néanmoins, aucune proposition n'est faite pour la création de messages entre objets à partir des éléments OWL. Pourtant, la définition de messages permet de représenter les interactions possibles entre objets.

La transformation entre langages peut se faire au travers d'une analyse syntaxique des spécifications décrivant le modèle de départ. Par exemple, Verdejo *et al.* [2005] effectuent une analyse syntaxique de documents RDF pour en extraire les informations à traduire en spécifications Maude. Lomuscio et Solanki [2009] proposent la transformation d'une ontologie OWL-S vers le langage ISPL (Interpreted System Programming Language), un langage d'entrée pour le modèle checker MCMAS [9] [Lomuscio et Raimondi, 2006]. Pour cela ils développent un compilateur qui implémente des règles de transformation syntaxiques pour générer automatiquement du code ISPL. Ces approches sont de la compilation au sens traduction d'un langage source à un langage

9. MCMAS est un model checker symbolique adapté à la vérification des systèmes multi-agents (SMA).

d'entrée. Cependant, les syntaxes des langages d'ontologies ont tendance à évoluer assez vite, l'utilisation de compilateur dans un processus de transformation entre deux langages, n'est donc pas une solution à long terme. Une majorité d'approches ont plus fréquemment recours à de la méta-modélisation [Anastasakis *et al.*, 2007; Huang *et al.*, 2009; Duràn et Roldàn, 2012; Gueffaz, 2012; Song *et al.*, 2012; Chama *et al.*, 2013]. À l'exemple de Huang *et al.* [2009] qui utilisent Maude afin de construire un modèle mathématique bien défini d'un système décrit en OWL-S. La méthode de transformation proposée consiste à définir une abstraction des sous-ensembles du langage OWL-S, puis à définir pour chaque sous-ensemble, la syntaxe et la sémantique correspondantes en logique de réécriture. Gueffaz [2012] dans son travail de thèse, investigue la vérification formelle de graphes sémantiques RDF à l'aide de techniques de model-checking. Pour cela, il propose une approche de transformation et de vérification nommée *ScaleSem*. Cette approche est fondée sur une transformation de graphe et a donné lieu à deux outils : RDF2SPIN [Gueffaz *et al.*, 2011b] qui transforme des graphes RDF en langage Promela pour une vérification à l'aide du model-checker SPIN [Holzmann, 2003]. et *RDF2NuSMV* [Gueffaz *et al.*, 2011a] qui transforme des graphes RDF en langage *NuSMV* [Cimatti *et al.*, 2000]. L'approche *ScaleSem* consiste à explorer l'ensemble du graphe RDF pour en obtenir tous les triplets, puis déterminer une racine globale pour le graphe et donner une représentation du graphe résultat en modèle écrit dans le langage Promela ou le langage NuSMV. Enfin, les propriétés du modèle sont vérifiées avec l'un des deux model-checkers SPIN ou NuSMV. L'outil OWL2Alloy [Song *et al.*, 2012] fait appel à l'API *Jena*, une plateforme RDF qui supporte OWL. L'API *Jena* permet d'analyser et manipuler les ontologies. L'objectif de OWL2Alloy est de transformer le niveau terminologique d'une ontologie OWL, c'est-à-dire ces concepts, propriétés et axiomes en un modèle Alloy afin d'y appliquer le raisonnement de l'analyseur Alloy. *OWL2Alloy* prend comme entrée les concepts, propriétés et axiomes de l'ontologie obtenus à l'aide de l'API *Jena*, puis génère à partir de ces éléments ontologiques les spécifications textuelles du modèle Alloy. Notre approche de transformation fait appel à un processus similaire, l'idée étant de prendre comme entrée du processus de génération du modèle formel un ensemble d'éléments provenant de l'ontologie. Néanmoins nous allons plus loin, car notre traduction implique en plus les individus, les instances de propriétés ainsi que les règles de comportement. En effet, contrairement à la majorité des approches de traduction de modèle, notre objectif n'est pas la traduction systématique de l'ensemble de l'ontologie OWL vers un langage de spécifications formelles mais seulement le sous-ensemble d'éléments

nécessaires à la représentation et à la vérification des instances de *Règle-Utilisateur* en une spécification Maude.

Dans le chapitre 3, nous avons donné une description formelle du modèle ontologique et décrit un ensemble de fonctions qui permettent de manipuler ce modèle. Dans ce chapitre nous donnons une description formelle du modèle Maude puis nous comparons les deux modèles pour proposer une transformation du modèle ontologique vers Maude. Cette approche a l'avantage de se passer à un méta-niveau qui ne tient pas compte des syntaxes respectives des langages utilisés par les deux modèles, car celles-ci peuvent exister sous différentes variantes et évoluer assez rapidement.

4.3 ÉLÉMENTS ONTOLOGIQUES À TRADUIRE

Il est important de rappeler que notre objectif n'est pas de traduire l'ensemble de l'ontologie en Maude, mais de dériver du modèle OWL les spécifications Maude suffisantes à la vérification du comportement dynamique du système modélisé. Ce comportement est défini majoritairement par les règles utilisateur identifiées à partir de l'analyse des spécifications d'exigences. Ces règles sont représentées dans l'ontologie par les individus du concept *Règle-utilisateur*. Les éléments à traduire sont donc les règles de comportement ainsi que l'ensemble des éléments nécessaires à leur définition, à savoir : les concepts, les propriétés et leurs instances. Dans notre approche plusieurs raisonnements ont été effectués sur l'ontologie avant sa transformation vers le langage de spécifications formelles. De la sorte, la cohérence de l'ontologie est avérée et l'ensemble des inférences a déjà pu être calculée. Il n'est donc pas utile de considérer les axiomes terminologiques autres que ceux décrivant les concepts et propriétés dans le processus de transformation de l'ontologie.

4.4 LE SYSTÈME MAUDE

4.4.1 *Fondement de Maude : logique de réécriture*

Maude [10] 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. 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, c'est-à-dire

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

des types (sortes) et opérateurs et $\mathcal{E}$ est l'ensemble des axiomes équationnels. Les sortes (*sort*) permettent d'introduire les types des objets, constantes et variables manipulés par Maude.

sort nom-sort .

La déclaration d'un opérateur donne les sortes des arguments et du résultat. Les opérateurs sont introduits par le mot clé op, un nom, un ensemble d'arguments (*sorts*) et un résultat (*sort*).

op nom-op : T_1 ... $T_n \rightarrow T_m$.

Lorsqu'ils n'ont pas d'argument, les opérateurs déclarent des constantes.

op nom-op : $\rightarrow T_m$. (Constante du type T_m)

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 $R : t \rightarrow t'$ si C avec t et t' des termes construits sur Σ pouvant contenir des constantes ou des variables. Ainsi toute instance du terme t, si la condition C est satisfaite modulo les mêmes substitutions de variables, est remplacé par l'instance correspondante du membre droit t'. Cette réécriture peut se faire sur un terme complet u si le terme complet est une instance de t ou seulement sur un sous-terme de u. Notons aussi que l'application des règles de réécriture est non déterministe : il peut y avoir plusieurs règles de réécriture qui peuvent s'appliquer au même terme, auquel cas Maude en choisit une.

4.4.2 *Structure*

Nous décrivons ici brièvement la structure d'une spécification Maude en nous concentrant essentiellement sur les éléments qui pourront être impliqués dans notre processus de transformation. Pour une description plus approfondie de Maude, le lecteur peut se référer à son manuel [Clavel *et al.*, 2011]. En Maude, les spécifications sont structurées en modules de tailles relativement petites pour faciliter la compréhension des gros systèmes, augmenter la ré-utilisabilité et localiser les effets des modifications du système. Maude définit trois types de modules :

1. module fonctionnel : y sont définis les types de données et les opérations associées au moyen de théories équationnelles (signature et équations entre termes). Un tel module est déclaré avec la syntaxe fmod Nom-du-module ... endfm.

2. module système : permet de définir le comportement dynamique d'un système, en augmentant le module fonctionnel par des règles de réécriture. Un tel module est déclaré avec la syntaxe mod Nom-du-module ... endm.
3. module orienté-objet : offre une syntaxe orienté objet plus adaptée à la définition de systèmes concurrents. Ces modules peuvent se réduire entièrement à des modules système[11]. Ils sont déclarés avec la syntaxe omod Nom-du-module ... endom.

Les modules orienté-objet sont ceux qui se prêtent le mieux à l'usage que nous recherchons. Nous nous intéressons dans la suite à la transformation possible de l'ontologie en un module orienté-objet.

Module orienté-objet

Les modules orienté-objet permettent de définir des systèmes concurrents à l'aide d'ensembles d'objets ayant chacun une identité unique, et d'un mécanisme de communication fondé sur le passage de messages entre objets. L'état d'un système concurrent, appelé configuration, est défini comme un multi-ensemble d'objets et de messages. Au sein d'un module orienté-objet, les objets sont définis par des classes.

SIGNATURES, VARIABLES En Maude, les types et les constructeurs permettant la modélisation des systèmes à base d'objets sont décrits par le module pré-défini CONFIGURATION (cf. Figure 18). Dans cette définition, les sortes Object et Msg sont laissées abstraites, elles seront définies, en ce qui nous concerne, dans le module objet résultant de notre transformation. La dernière sorte (Configuration) dans la Figure 18 subsume Object et Msg. Configuration est définie par l'opérateur "__" comme une concaténation de Configuration. Les indications à droite de cet opérateur de concaténation précisent que l'ordre d'apparition des objets et messages dans une configuration n'est pas significatif.

CLASSES Une classe définit un type d'objet portant sur les mêmes attributs et partageant les mêmes comportements. Une classe est représentée par une expression de la forme :

$$\text{class } C \mid a_1 : s_1, ..., a_n : s_n.$$

Avec *class* un mot-clé, C le nom de la classe, suivi d'une barre '|', suivie d'une liste de déclarations d'attributs a_i séparés par des virgules. Les s_i correspondent aux sortes (types) des attributs.

11. Les modules orienté-objet ne sont que du sucre syntaxique, avant l'exécution, ils sont transformés en modules système.

```
mod CONFIGURATION is
- - - Déclaration des sortes (types) de base des systèmes objets - - -
sorts Object Msg Configuration .
- - - Configuration est un multi-ensemble d'objets et de messages - - -
subsort Object Msg < Configuration .
op none : → Configuration [ctor] .
op __ : Configuration Configuration
              → Configuration [ctor config assoc comm id : none] .
```

FIGURE 18: Module prédéfini *CONFIGURATION*

OBJETS Un objet est représenté par un terme de la forme :

$$< O : C | a_1 : v_1, ..., a_n : v_n >$$

avec O l'identifiant (constante déclarée de sorte Oid) de l'objet, C sa classe, les a_i, pour i de 1 à n, sont les noms d'attributs de l'objet et les v_i les valeurs d'attribut.

MESSAGES Les messages n'ont pas une forme syntaxique fixe, leur syntaxe est définie par l'utilisateur

lors de la déclaration du message dont la forme est donnée ci-dessous. La seule hypothèse faite par Maude, est que le premier argument d'un message est l'identifiant de l'objet destinataire.

$$\text{msg } Mes : Oid, T_1, ..., T_n \rightarrow Msg.$$

avec msg un mot clé, Mes le nom du message, Oid le type de l'objet destinataire et les T_i les types des arguments des messages qui peuvent être des objets (Oid) ou tout autre type défini.

CONFIGURATIONS Une configuration est un multi-ensemble d'objets et de messages avec comme forme générale :

$$Ob_1 \ ... \ Ob_m \ Mes_1 \ ... \ Mes_n$$

avec Ob_1 , ... , Ob_m des objets et Mes_1 , ... , Mes_n des messages, l'ordre n'est pas significatif au sein d'une configuration.

ÉQUATIONS Maude permet de définir des axiomes encore appelés équations, de la forme $t = t'$. Ces équations correspondent à des règles de simplification. Dans la suite, nous faisons une utilisation très restreinte de ces équations qui permettent de définir des configurations.

$$\text{eq } Conf = Ob_1 \ ... \ Ob_m \ Mes_1 \ ... \ Mes_n.$$

avec eq mot clé pour équation, Conf le nom de la configuration, Ob_i les objets contenus dans la configuration, Mes_i les messages conte-

nus dans la configuration. Conf est déclarée au préalable comme une constante de type Configuration.

$$\text{op } Conf \rightarrow Configuration.$$

Définir des configurations permet de faire appel à un ensemble d'objets et de messages sans les réécrire systématiquement. Cela s'avère très utile dans la définition de propriétés à satisfaire et d'états à atteindre ou ne pas atteindre.

RÈGLES DE RÉÉCRITURE Les règles de réécriture permettent de réécrire toute instance du membre gauche de la règle en l'instance correspondante du membre droit. Ces règles peuvent être inconditionnelles (noté rl) ou conditionnelles (noté crl). Dans ce dernier cas leur application dépend de la satisfaction d'une condition.

$$\text{rl [Label] : } Configuration_1 \Rightarrow Configuration_2$$

$$\text{crl [Label] : } Configuration_1 \Rightarrow Configuration_2 \text{ if } Condition_1 \wedge ... \wedge Condition_n$$

Dans une telle règle de réécriture, les objets du membre gauche peuvent être réécrits, ils apparaissent alors dans le membre droit avec des valeurs d'attributs actualisées. Néanmoins, l'ensemble des messages du membre gauche n'est pas réécrit. Une réécriture peut donner lieu à la création de nouveaux objets et messages. Les règles de réécriture pour les systèmes orienté-objets sont de la forme :

$$\text{rl } Ob_1 \ ... \ Ob_k \ Mes_1 \ ... \ Mes_n \Rightarrow Ob_1' \ ... \ Ob_i' \ Ob_1'' \ ... \ Ob_j'' \ Mes_1' \ ... \ Mes_m' \ .$$

$$\text{crl } Ob_1 \ ... \ Ob_k \ Mes_1 \ ... \ Mes_n \Rightarrow Ob_1' \ ... \ Ob_i' \ Ob_1'' \ ... \ Ob_j'' \ Mes_1' \ ... \ Mes_m' \text{ if } (t_1 = t_1') \wedge ... \wedge (t_n = t_n') \ .$$

avec $Ob_1' \ ... \ Ob_i'$ les versions actualisées des objets $Ob_1 \ ... \ Ob_k$ pour $i \leqslant k$, $Ob_1'' \ ... \ Ob_j''$ les nouveaux objets et $Mes_1' \ ... \ Mes_m'$ sont de nouveaux messages.

4.4.3 *Mécanismes de vérification*

Les éléments principaux de Maude sont les équations et les règles. Quoiqu'utilisées à des fins différentes, elles s'utilisent de la même façon : les éléments du membre gauche sont remplacés par les éléments du membre droit. Ainsi les ensembles d'objets et de messages évoluent en fonction des règles de réécriture. Celles-ci décrivent les interactions possibles entre les objets. Les modules Maude sont exécutables et leur exécution fait appel essentiellement à trois types de commandes :

- la commande *reduce* permet la simplification de termes en utilisant les équations d'un module, jusqu'à l'obtention d'une forme normale, c'est-à-dire un terme qui ne peut plus être simplifié.
- la commande *rewrite* (rew) permet la réécriture d'un terme, ici une configuration, en utilisant les règles de réécriture. Elle consiste à effectuer, à partir d'un état initial, une à plusieurs séquences de réécriture et ce jusqu'à ce que plus aucune réécriture ne soit possible.
- la commande *search* permet d'explorer toutes les séquences possibles de réécriture (en utilisant les règles de réécriture) à partir d'un état initial donné. Elle examine chaque séquence de réécriture, en vue de vérifier la satisfaction d'une certaine formule. La recherche s'achève lorsque toutes les séquences possibles ont été appliquées ou lorsque l'on a trouvé une configuration qui satisfait la formule. La vérification du modèle ne se termine pas si l'espace d'états atteignables est infini et que la configuration recherchée n'est pas accessible à partir de l'état initial. Le mécanisme de recherche repose sur l'application des mécanismes de simplification et de réécriture. La stratégie de Maude impose que toutes les simplifications équationnelles soient appliquées avant chaque application des règles de réécriture.

Notre objectif est de pouvoir explorer l'ensemble des configurations possibles du système, représentant le résultat des applications de règles de comportement modélisées, afin de vérifier l'atteinte d'états désirés ou non. La commande qui nous intéresse alors est *search*. Nous détaillons ci-dessous sa forme et son utilisation :

```
search [ n, m ] in nom-du-module : Configuration-1 ⇒(mod)
                Configuration-2 such that Condition .
```

- **n** (optionnel) nombre de solutions désirées ;
- **m** (optionnel) profondeur maximum de la recherche (réécriture) ;
- **nom-du-module** (optionnel) ;
- **Configuration-1** : configuration de départ ;
- **Configuration-2** : l'état qui doit être atteint ;
- **SearchArrow** (⇒) : indique le sens de réécriture ;
- **mod** : détermine la forme de la séquence de réécriture allant de Configuration-1 à Configuration-2 :
  - ⇒1 : une seule étape ;
  - ⇒+ une ou plusieurs étapes ;
  - ⇒* aucune, une ou plusieurs étapes ;
  - ⇒! indique que seuls les états finaux en forme normale sont autorisés, c'est-à-dire les états qui ne peuvent être réécrits ;
- **Condition** (optionnelle) : indique une propriété qui doit être satisfaite par l'état atteint ; la forme syntaxique de la condition est la même que pour les conditions des règles conditionnelles.

4.5 VÉRIFICATION DU MODÈLE SOUS MAUDE

Nous avons énoncé en section 1.8.3 l'ensemble des vérifications que nous souhaitons appliquer aux règles de comportement issues de l'analyse des spécifications d'exigences. En section 3.2 nous avons montré que OWL permet la vérification de l'applicabilité et de la cohérence des règles utilisateur et que seule la vérification de la conformité des règles reste à effectuer ; car celle-ci nécessite l'exploration des états possibles du système ce que ne permet pas OWL. Dans cette section, nous décrivons donc la vérification de la conformité du comportement système spécifié par les règles utilisateurs et ce par rapport à différentes propriétés. En règle générale, ces propriétés de « conformité » définissent les états ou configurations du système que l'on peut souhaiter atteindre ou ne jamais atteindre. Ces propriétés ne sont pas données par l'utilisateur du système mais par un expert qui connaît les contraintes du domaine à respecter. Les propriétés de conformité les plus courantes et les plus utilisées sont les propriétés de sécurité nommées *invariant*. Un invariant sur les états d'un système est un prédicat qui est vrai dans tous les états du système qui sont accessibles à partir d'une certaine classe d'états initiaux [Rusu et Clavel, 2009]. Ces propriétés indiquent que quelque chose de contre-indiqué ne devrait jamais arriver.

Pour l'ensemble des vérifications, nous faisons appel à la commande *search* et à ses différentes modalités. L'utilisation de cette commande demande la donnée d'une configuration initiale. Dans notre cas, cette configuration peut contenir l'ensemble ou un sous-ensemble des objets du système ainsi que les messages décrivant leurs interactions. Pour plus de commodité, nous créons une configuration pour chaque ensemble d'entités [12] d'un même concept. Ci-dessous, la configuration *Conf-*C_x est déclarée à la première ligne, puis définie par une équation à la seconde ligne.

$$op\ Configuration\ Conf\text{-}C_x\ .$$
$$eq\ Conf\text{-}C_x = i_{1C_x}\ ...\ i_{nC_x}$$

De la même façon, les messages entre couples d'entités (d'objets) sont regroupés dans la configuration *Conf-Mes*. Ci-dessous, la configuration *Conf-Mes* est déclarée à la première ligne, puis définie par une équation à la seconde ligne.

$$op\ Configuration\ Conf\text{-}Mes\ .$$
$$eq\ Conf\text{-}Mes = \mathrm{Mes}_1\ ...\ \mathrm{Mes}_m$$

Ainsi, l'expert peut composer selon les vérifications à effectuer les différentes configurations système initiales qui serviront comme point de départ au processus de vérification et sur lesquelles s'appliqueront

12. individus de type référent individuel, i.e. individus du concept *Composant*

les règles de réécriture issues de notre modèle ontologique. Ces configurations initiales sont de la forme :

op Configuration Conf-$\mathtt{init}_i$.
eq Conf-$\mathtt{init}_i$ = *Conf-*C_1 ... *Conf-*C_n *Conf-Mes* .

L'exploration du modèle permet de vérifier aussi bien l'atteignabilité de certains états que la non-atteignabilité de certains autres. Valider l'atteignabilité d'une configuration consiste à vérifier la valeur d'attributs des objets ou simplement l'existence d'objets ou de messages. Dans le premier cas, il s'agit de vérifier que pour certains objets des valeurs d'attributs particulières peuvent être atteintes ou inversement que certaines valeurs d'attributs ne sont jamais atteintes. Ci-dessous est décrit un exemple de vérification relevant du premier cas. La recherche a pour configuration initiale *Conf-*$\mathtt{init}_i$. La configuration recherchée est une configuration qui contiendrait deux objets O_1 et O_k dont les valeurs d'attributs v_{11} et v_{k1} seraient égales ($v_{11} = v_{k1}$).

search *Conf-*$\mathtt{init}_i \Rightarrow * C$: Configuration $< O_1 : C_1 | a_{11} : v_{11}, \ldots , a_{1n} : v_{1n} > \ldots < O_k : C_k | a_{k1} : v_{k1}, \ldots , a_{km} : v_{km} >$ such that $(v_{11} == v_{k1})$.

Ce type de recherche peut être utilisé pour vérifier qu'une certaine quantité n'est jamais atteinte : par exemple, ne pas dépasser une quantité d'énergie consommée à un instant t ou ne pas dépasser un budget autorisé.

Dans le second cas, il s'agit de vérifier s'il existe une configuration dans laquelle certains messages sont créés ou si certains messages ne sont jamais créés. Ci-dessous est décrit un exemple de vérification relevant du premier cas. La recherche a pour configuration initiale *Conf-*$\mathtt{init}_j$. La configuration recherchée est une configuration qui contiendrait les objets O_1 ... O_n et les messages Mes_1 ... Mes_m.

search *Conf-*$\mathtt{init}_j \Rightarrow * C$: Configuration $O_1 ... O_n$ $Mes_1 ... Mes_m$

Ce type de recherche peut être utilisé pour vérifier qu'il existe une séquence de réécriture qui mènera à la création de certains messages, par exemple qu'un *actionneur* pourra à un moment envoyer un message *allumer* à un *appareil* ou encore pour vérifier que deux messages (tels que *allumer* et *éteindre*) ne peuvent être envoyés simultanément à un même appareil.

La vérification proposée par la commande *search* a l'avantage d'explorer le modèle de manière complètement automatique. Lorsqu'une erreur est trouvée dans le processus de model checking, un contre-exemple est toujours produit. Ce contre-exemple sert à comprendre d'où vient l'erreur et la corriger.

Un autre usage intéressant de la commande *search*, que nous préconisons dans notre approche, porte sur la validation de règles de comportement. Cette validation se fait par l'animation du modèle,

qui permet de suivre les comportements possibles du système sur un certain nombre d'étapes. Plus précisément, il s'agit d'observer l'arbre d'exécution des règles de comportement ainsi que les configurations du système obtenues. L'exploration de cet arbre d'exécution peut faire apparaître des comportements non conformes (inattendus ou non désirés) aux attentes de l'utilisateur. L'examen de chaque étape d'application d'une règle peut alors permettre de comprendre et de remédier aux causes de ces comportements non conformes ou au contraire de valider la conformité du comportement décrit par les règles utilisateur.

Comme il a été indiqué en section 3.3.6.3, chaque entité de l'ontologie se voit attribuer le numéro de phrase et le numéro de nœud syntaxique du terme qui y fait référence dans les spécifications d'exigences. La traduction que l'on propose a pour objectif de préserver ces connaissances. Riche de ces connaissances, le modèle Maude permet alors de maintenir un lien entre les spécifications d'exigences en langage naturel et les spécifications formelles. L'expert peut ainsi passer directement des contre-exemples et entités qui posent problème à leur dénotation dans les textes et naviguer de l'un à l'autre.

4.6 CORRESPONDANCE ENTRE LE MODÈLE OWL ET LE MODÈLE MAUDE

Dans cette section, nous mettons en relief et décrivons les correspondances qui existent entre les éléments du modèle ontologique que nous avons proposé (cf. section 3.1) et les éléments du module orienté objet de Maude. Les éléments ontologiques décrits sont ceux qui participeront à la représentation de l'évolution de l'état du système. Ces éléments correspondent aux instances du concept *Règle-utilisateur* issues de l'analyse des spécifications d'exigences ainsi que les éléments nécessaires à leur définition et exécution, à savoir : les concepts *Composant* et *Type*, les propriétés (relations et attributs), ainsi que les individus et leurs valeurs de propriété.

Le tableau 19 illustre les différentes correspondances relatives à notre contexte d'utilisation. Les objets Maude, de même que les individus (référents individuels) du concept *Composant* représentent des entités. Quant aux individus (référents génériques) du concept *Type*, ils représentent des valeurs d'attributs et seront donc représentés comme tels dans le modèle Maude. Les sous-concepts du concept *Composant* correspondent alors à des classes d'objets Maude. Quant aux sous-concepts du concept *Type*, ils correspondent à la sorte « objet » (*Oid*). Les messages Maude sont définis entre plusieurs objets. Les propriétés de type *relation* portent sur des couples d'individus du concept *Composant*. Les relations correspondent alors à des messages Maude

entre deux objets de la classe *Composant*. Les propriétés de type *attribut* lient une entité à une entité ou à une valeur de type la caractérisant. De manière similaire, les attributs de classe Maude lient un objet à une sorte. Les attributs OWL sont donc mis en correspondance avec les attributs de classe Maude. Enfin, les instances de *Règle-utilisateur* sont formées tout comme les règles de réécriture Maude d'un antécédent et d'un conséquent. Ces derniers portent dans les deux cas sur des entités, des relations, des attributs, des valeurs de types simples (littéraux) et des variables. Dans les règles de réécriture Maude, deux types d'éléments sont déclencheurs : les messages qui correspondent aux relations de l'ontologie et les valeurs d'attributs des objets. Lors de l'application d'une règle de réécriture, les messages de l'antécédent ne sont pas réécrits dans le conséquent et seuls les valeurs des attributs *dynamiques* (cf. section 3.1.1.1) peuvent évoluer.

On peut donc voir que le modèle ontologique que nous proposons permet une correspondance directe entre les éléments de notre ontologie et ceux du modèle orienté objet Maude.

Ontologie OWL	Module orienté-objet Maude
Individu du concept *Composant*	Objet
Individu du concept *Type*	Valeur d'attribut
Concept *Composant*	Classe
Concept *Type*	Sorte *Oid*
Propriété (relation)	Message
Propriété (attribut)	Attribut
Instance de *Règle-Utilisateur*	Règle de réécriture

FIGURE 19: Correspondance entre notre ontologie et Maude

4.7 TRANSFORMATION DU MODÈLE ONTOLOGIQUE EN MODULE ORIENTÉ-OBJET MAUDE

4.7.1 *Méthode de transformation*

Notre objectif est de mettre en correspondance les éléments de notre modèle ontologique avec ceux du modèle orienté objet Maude. En section 3.1.1, nous avons présenté une vision ensembliste du modèle ontologique $\mathbb{O} = < \mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}} >$. Nous exploitons cette vision, ainsi que les fonctions de manipulation d'ontologie décrites en section 3.1.3 pour outiller le passage entre le modèle ontologique et le modèle orienté objet Maude. Ce mode de transformation à 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.

Le processus de transformation de spécifications formelles est illustré par la figure 20. De manière similaire à l'approche proposée par Song *et al.* [2012] ce processus 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 (cf. tableau 1) implantées à l'aide des API Java *OWL-API* et *Jess*. L'objectif est de sélectionner les éléments ontologiques nécessaires à la création du modèle orienté objet Maude. La traduction du modèle ontologique en un modèle Maude est suivie par l'application de fonctions dites de *pretty-printing* 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. Dans la suite, nous définissons le modèle orienté objet Maude et décrivons le processus de transformation de notre modèle ontologique vers le modèle Maude ainsi que le code Maude généré à chaque application d'une fonction de *pretty-printing*.

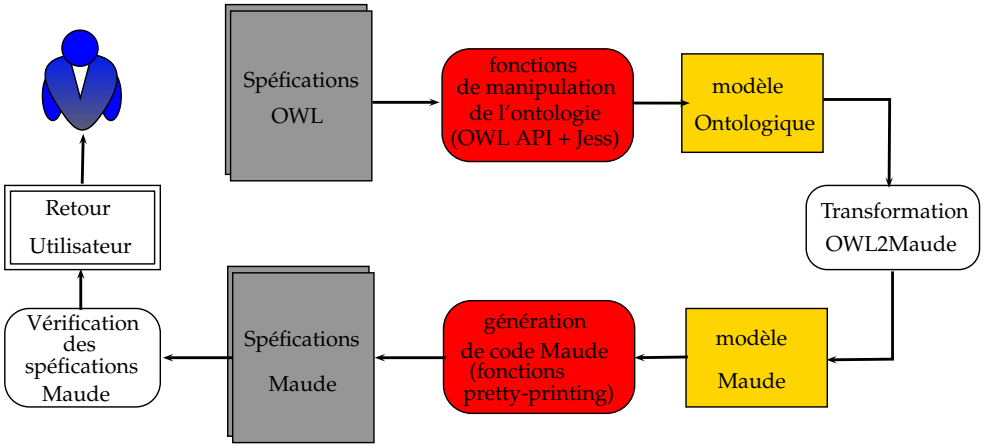


FIGURE 20: Transformation et vérification de l'ontologie OWL sous Maude

4.7.2 *Définition du modèle orienté objet Maude*

Comme l'illustre la figure 20, la génération de spécifications Maude passe d'abord par la transformation du modèle ontologique en un modèle Maude. Dans cette section nous donnons une définition de ce modèle qui est fondée sur le paradigme orienté objet de Maude.

Le modèle Maude orienté objet correspond à un tuple noté $<\mathcal{C},\mathcal{M},\Sigma,\mathcal{E},\mathcal{R}>$ où

– $\mathcal{C}$ contient l'ensemble des noms de classes avec pour chaque classe son ensemble de couples (attribut, type).

- $\mathcal{M}$ contient l'ensemble des noms de messages.
- Σ correspond à l'environnement de typage. Chaque élément (constante ou variable) y est associé à son type.
- $\mathcal{E}$ correspond à l'ensemble des équations représentant l'état du système (sa configuration) avec $\mathcal{E} = \mathcal{E}_{\mathcal{O}} \cup \mathcal{E}_{\mathcal{M}}$ tel que :
  - $\mathcal{E}_{\mathcal{O}}$: l'ensemble des couples configurations-objets.
  - $\mathcal{E}_{\mathcal{M}}$: l'ensemble des couples configurations-messages.
- $\mathcal{R}$ contient l'ensemble des règles de réécriture avec pour chaque règle son ensemble d'objets et de messages.

4.7.3 *Traduction du modèle ontologique en un modèle Maude*

L'ensemble des éléments du modèle OWL nécessaires à la vérification du comportement du système sont traduits en un modèle orienté objet Maude.

Pour cela, nous avons implémenté une fonction de traduction que nous appelons $\mathtt{Trad}_O$ qui a pour objectif de permettre la correspondance entre les deux modèles :

$$\mathtt{Trad}_O(\mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}}) = <\mathcal{C},\mathcal{M},\Sigma,\mathcal{E},\mathcal{R}>$$

La fonction de traduction $\mathtt{Trad}_O$ fait appel à 4 autres fonctions de traduction intermédiaires, $\mathtt{Trad}_C$, $\mathtt{Trad}_M$, $\mathtt{Trad}_E$ et $\mathtt{Trad}_R$, qui permettent de créer ou de compléter un ou plusieurs composants du modèle Maude. Chaque fonction de traduction intermédiaire permet de créer un sous-ensemble d'éléments Maude. Les arguments de ces fonctions correspondent aux ensembles d'éléments OWL nécessaires à la création de chacun des ensembles Maude. Dans la suite nous décrivons les quatre fonctions et leur mécanisme de transformation d'éléments OWL en éléments Maude.

La génération de la spécification Maude se fait par l'appel de la fonction *pp-génération-du-code-Maude*($\mathcal{C}$,$\mathcal{M}$,Σ,$\mathcal{E}$,$\mathcal{R}$,*Spec-Maude*) qui prend en entrée les éléments du modèle Maude obtenu par l'application de la fonction $\mathtt{Trad}_O$ ainsi que le document (*Spec-Maude*) de spécification Maude qui contiendra le code généré. Le processus de génération de la spécification Maude correspond au module de génération de code Maude illustré en figure 20. La fonction *pp-génération-du-code-Maude* fait appel à plusieurs fonctions intermédiaires (cf. Algorithme 7). Chacune de ces fonctions intermédiaires a pour rôle l'écriture d'un type d'ensemble du modèle Maude. Néanmoins ces fonctions intermédiaires de génération de code Maude (toutes préfixées par pp) ne sont pas décrites ici mais uniquement spécifiées par le code qu'elles génèrent. Ceci est dénoté par l'opérateur $\rightsquigarrow$ qui suit l'en-tête d'une fonction de génération de code encore dite fonction de *pretty-printing*. Pour plus de lisibilité la génération de code de ces fonctions de *pretty-*

printing est illustrée après chaque algorithme de fonction de traduction. En effet, la forme syntaxique qui y est précisée permet de faire plus facilement le lien avec les éléments de spécification Maude qui sont calculés par l'algorithme. L'opérateur ← qui apparaît dans l'algorithme 7 dénote l'écriture du code de chaque fonction intermédiaire dans le document (*Spec-Maude*) de spécification Maude donnée en entrée.

Input : $< \mathcal{C}, \mathcal{M}, \Sigma, \mathcal{E}, \mathcal{R} >$,*Spec-Maude*
Output : *Spec-Maude*
Spec-Maude ← *pp-declareClass*($\mathcal{C}$);
Spec-Maude ← *pp-declareMessage*($\mathcal{M}$);
Spec-Maude ← *pp-declareObject*(Σ);
Spec-Maude ← *pp-declareVariables*(Σ);
Spec-Maude ← *pp-declareObjectConfiguration*(Σ);
Spec-Maude ← *pp-createObjectConfiguration*($\mathcal{E}$);
Spec-Maude ← *pp-createMsgConfiguration*($\mathcal{E}$);
Spec-Maude ← *pp-createRules*($\mathcal{R}$);

Algorithme 7: Génération du code Maude

1) La fonction $\mathtt{Trad}_O$ *: génération du modèle Maude*

La fonction $\mathtt{Trad}_O$ prend en entrée l'ensemble des éléments du modèle ontologique. Elle a pour objectif de générer à partir de ceux-ci le modèle Maude, à partir duquel sera générée la spécification du module orienté objet correspondant.

L'exécution de la fonction $\mathtt{Trad}_O$ est décrit par l'algorithme 8.

Input : $\mathbb{C}, \mathbb{P}, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}}$
Output : $< \mathcal{C}, \mathcal{M}, \Sigma, \mathcal{E}, \mathcal{R} >$
$\mathbb{C}_C \leftarrow$ *getConceptSubClasses*($\mathbb{A}$,*Composant*);
$\mathbb{C}_T \leftarrow$ *getConceptSubClasses*($\mathbb{A}$,*Type*);
$\mathbb{C}_{RU} \leftarrow$ *getSubConcepts*(*Règle-Utilisateur*,$\mathbb{A}$);
$\mathbb{I}_{RU} \leftarrow$ *getConceptIndividuals*($\mathbb{C}_{RU}$, $\mathfrak{I}^{\mathbb{C}}$);
$\mathbb{P}_R \leftarrow$ *getOntologyRelations*($\mathbb{A}$);
$\mathcal{C} \leftarrow \mathtt{Trad}_C(\mathbb{C}_C, \mathbb{C}_T, \mathbb{A})$;
$\mathcal{M} \leftarrow \mathtt{Trad}_M(\mathbb{P}_R)$;
$<\mathcal{E}, \Sigma_0> \leftarrow \mathtt{Trad}_E(\mathbb{C}_C, \mathbb{P}_R, \mathbb{A}, \mathbb{I}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}})$;
$<\mathcal{R}, \Sigma> \leftarrow \mathtt{Trad}_R(\mathbb{I}_{RU}, \mathfrak{I}^{\mathbb{P}}, \mathbb{P}_R, \mathbb{A}, \Sigma_0)$;

Algorithme 8: Génération du module orienté objet Maude

2) La fonction $\mathtt{Trad}_C$ *: déclaration des classes Maude*

La fonction $\mathtt{Trad}_C$ a pour objectif de créer (déclarer) l'ensemble des classes Maude correspondant aux concepts de l'ontologie. Pour cela cette fonction prend en arguments l'ensemble des concepts de *Composant* ($\mathbb{C}_C$) et de *Type* ($\mathbb{C}_T$), ainsi que l'ensemble des axiomes terminologiques ($\mathbb{A}$). Les connaissances nécessaires à la création des classes sont les noms des concepts, leurs attributs ainsi que les types associés à leurs valeurs d'attributs (valeurs d'image). L'algorithme 9 détaille le processus de création des classes.

Input : $\mathbb{C}_C, \mathbb{C}_T, \mathbb{A}$
Output : $\mathcal{C}$
$\mathcal{C} \leftarrow \emptyset$;
for *each* C *in* $\mathbb{C}_C \cup \mathbb{C}_T$ **do**
  $P_A \leftarrow \mathtt{getConceptAttributes}(C, \mathbb{A})$;
  Attributes-Types $\leftarrow \emptyset$;
  for *each* $\mathtt{att}$ *in* P_A **do**
    $t \leftarrow \mathtt{getRange}(\mathtt{att}, \mathbb{A})$;
    Attributes-Types $\leftarrow$ *Attributes-Types* $\cup \{(\mathtt{att}, t)\}$;
  end
  $\mathcal{C} \leftarrow \mathcal{C} \cup$ {(C, Attributes-Types)};
end

Algorithme 9: Déclaration des classes Maude

Génération de code pour la déclaration des classes

pp-declareClass($\mathcal{C}$)

$\rightsquigarrow$ *class* C_1 | $\mathtt{att}_{11} : t_{11}, ..., \mathtt{att}_{1n} : t_n$.

...

class C_m | $\mathtt{att}_{m1} : t_{m1}, ..., \mathtt{att}_{mn} : t_{mn}$.

avec $\mathcal{C} = \{(C_1, \{(\mathtt{att}_{11}, t_{11}), \ldots, (\mathtt{att}_{1n}, t_n)\}), ..., (C_m, \{(\mathtt{att}_{m1}, t_{m1}), \ldots, (\mathtt{att}_{mn}, t_{mn})\})\}$

3) La fonction $\mathtt{Trad}_M$ *: déclaration des messages Maude*

La fonction $\mathtt{Trad}_M$ a pour objectif de créer (déclarer) l'ensemble des messages Maude correspondant aux relations entre concepts *Composant* de l'ontologie. Les relations d'une ontologie sont des prédicats binaires définis sur des ensembles d'individus (objets au sens Maude). Il en résulte que les messages Maude seront nécessairement définis entre couples d'objets (Oid). La création d'un message ne requiert alors que le nom de la propriété de type relation. La fonction $\mathtt{Trad}_M$ prend en argument l'ensemble des axiomes terminologiques $\mathbb{A}$ à par-

tir desquels l'ensemble $\mathbb{P}_R$ des relations de l'ontologie peut être obtenu. Ce processus est illustré par l'Algorithme 10.

Input : $\mathbb{P}_R$
Output : $\mathcal{M}$
$\mathcal{M} \leftarrow \emptyset$;
for *each* P_R *in* $\mathbb{P}_R$ **do**
  //P_R correspond à un nom de message
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{P_R\}$;
end

Algorithme 10: Déclaration des messages Maude

Génération de code pour la déclaration des messages

pp-declareMessage($\mathcal{M}$)
$\rightsquigarrow$ *msg* Msg_1 : *Oid Oid* $\rightarrow$ *Msg* .
...
msg Msg_n : *Oid Oid* $\rightarrow$ *Msg* .
avec $\mathcal{M} = \{Msg_1, ..., Msg_n\}$

4) La fonction $Trad_E$ *: déclaration et création des configurations*

La fonction $Trad_E$ a pour objectif de créer l'ensemble des configurations qui permettra de représenter l'état du système. La création de ces configurations revient à la définition d'équations reliant une configuration définie (préalablement déclarée) à un ensemble d'objets ou de messages. Dans notre approche de vérification du modèle Maude (cf. 4.5), les objets correspondant aux individus de l'ontologie sont groupés dans des configurations représentant chacune un type de concept. Les messages représentant les relations entre individus de l'ontologie sont groupés dans une configuration séparée.

La fonction $Trad_E$ se décline en deux fonctions distinctes, l'une permettant la déclaration et la création des configurations d'objets ($Trad_{E_O}$), l'autre, la configuration de messages entre objets ($Trad_{E_M}$).

4.1) La fonction $Trad_{E_O}$ *: déclaration et création des configurations d'objets*

La fonction $Trad_{E_O}$ prend en entrée un ensemble des concepts *Composant* ($\mathbb{C}_C$), l'ensemble des axiomes terminologiques ainsi que la fonction $\mathfrak{I}^C$ qui permet d'obtenir l'ensemble des individus du concept *Composant* ($\mathfrak{I}^C(Composant) \subset \mathbb{I}$) et la fonction $\mathfrak{I}^P$. Pour chaque individu de cet ensemble, une déclaration de l'objet Maude correspondant est faite, puis l'objet est ajouté à l'environnement de typage Σ_0. Puis l'objet est créé à partir du nom de l'individu, de ses attributs et de ses valeurs d'attributs et ajouté à l'ensemble des configurations $\mathcal{E}_0$. Le

résultat de cette fonction est la déclaration et la création de configurations d'objets pour chaque ensemble d'individus d'un concept. Ce processus est décrit par l'Algorithme 11.

Input : $\mathbb{C}_C, \mathbb{A}, \mathfrak{I}^{\mathbb{C}}, \mathfrak{I}^{\mathbb{P}}$
Output : Σ_0,$\mathcal{E}_{\mathcal{O}}$
$\Sigma_0 \leftarrow \emptyset$;
$\mathcal{E}_{\mathcal{O}} \leftarrow \emptyset$;
for *each* C *in* $\mathbb{C}_C$ **do**
  $\Sigma_0 \leftarrow \Sigma_0 \cup \{(C, \mathtt{Configuration})\}$;
  $A \leftarrow$ *getConceptAttributes*$(C, \mathbb{A})$;
  $I_C \leftarrow$ *getConceptIndividuals*$(\mathbb{C}, \mathfrak{I}^{\mathbb{C}})$;
  for *each* i_C *in* I_C **do**
    Attributes-Values $\leftarrow \emptyset$;
    for *each* $\mathtt{att}$ *in* A **do**
      $v \leftarrow$ *getRangeValue*$(i_C, \mathtt{att}, \mathfrak{I}^{\mathbb{P}})$;
      Attributes-Values $\leftarrow$ *Attributes-Values* $\cup \{(\mathtt{att}, v)\}$;
    end
    $\Sigma_0 \leftarrow \Sigma_0 \cup \{(i_C, \mathtt{Oid})\}$;
    $\mathcal{E}_{\mathcal{O}} \leftarrow \mathcal{E}_{\mathcal{O}} \cup \{(C, i_C,$ *Attributes-Values*$)\}$;
  end
end

Algorithme 11: Création des configurations d'objets

Génération de code pour la déclaration de configurations d'objets

pp-declareObjectConfiguration(Σ_0)
$\rightsquigarrow$ *op Config-*$C_1 \rightarrow$ *Configuration* .
...
op Config-$C_n \rightarrow$ *Configuration* .

avec $\Sigma_0 = \{\{(C_1, \mathtt{Configuration}), ..., (C_n, \mathtt{Configuration})\}, ...\}$

Génération de code pour la déclaration d'objets

pp-declareObject(Σ_0)
$\rightsquigarrow$ *op* $i_{C_1} \rightarrow$ *Oid* .
...
op $i_{C_n} \rightarrow$ *Oid* .

avec $\Sigma_0 = \{\{(i_{C_1}, \mathtt{Oid}), ..., (i_{C_n}, \mathtt{Oid})\}, ...\}$

Génération de code pour la création de configurations d'objets

$$\textit{pp-createObjectConfiguration}(\mathcal{E}_{\mathcal{O}})$$
$$\rightsquigarrow \textit{eq Config-}C_1 = < i_{C_i} : C_i | att_{i1} : v_{i1}, ..., att_{in} : v_{in} > ... < i_{C_u} : C_u | att_{u1} : v_{u1}, ..., att_{un} : v_{un} > .$$
$$...$$
$$\textit{eq Config-}C_z = < i_{C_j} : C_j | att_{1j} : v_{1j}, ..., att_{jm} : v_{1m} > ... < i_{C_t} : C_t | att_{1t} : v_{1t}, ..., att_{mt} : v_{mt} > .$$
$$\textit{avec } \mathcal{E}_{\mathcal{O}} = \{(i_{C_1}, C_1, \{(att_{11}, t_{11}), ..., (att_{1n}, t_n)\}), ..., ((i_{C_m}, C_m, \{(att_{m1}, t_{m1}), ..., (att_{mn}, t_{mn})\})\}$$

4.2) La fonction $\mathtt{Trad}_{E_M}$: déclaration et création des configurations de messages

La fonction $\mathtt{Trad}_{E_M}$ prend en entrée l'ensemble des propriétés de type relation ($\mathbb{P}_R$), la fonction $\mathfrak{I}^{\mathbb{P}}$ qui permet de d'obtenir l'ensemble des instances de relation $\mathfrak{I}^{\mathbb{P}}(\mathbb{P}_R)$ et l'ensemble des configurations d'objets $\mathcal{E}$, résultat de la fonction de traduction $\mathtt{Trad}_{E_O}$. La fonction crée pour chaque instance de relation de ce dernier ensemble, un message qui contient le nom de l'instance de propriété ainsi que les objets (individus) qui lui sont associés. Par la suite, les messages créés sont regroupés dans la configuration *Config-Msg*. Le résultat de cette fonction est la déclaration et la création d'une configuration de messages représentant l'ensemble des relations de l'ontologie $\mathbb{P}_R$. Ce processus est décrit par l'Algorithme 12.

Input : $\mathbb{P}_R, \mathfrak{I}^{\mathbb{P}}, \mathcal{E}_{\mathcal{O}}$
Output : $\mathcal{E}$
$\mathcal{E}_{\mathcal{M}} \leftarrow \emptyset$;
for *each* P_R *in* $\mathbb{P}_R$ **do**
  $I_{P_R} \leftarrow \mathtt{getPropertyInstances}(P_R, \mathfrak{I}^{\mathbb{P}})$;
  for *each* i_{P_R} *in* I_R **do**
    //i_{P_R} correspond à un nom de message
    $i_D \leftarrow$ *getDomainValue*(i_{P_R},P_R,$\mathfrak{I}^{\mathbb{P}}$);
    $i_R \leftarrow$ *getRangeValue*(i_{P_R},P_R,$\mathfrak{I}^{\mathbb{P}}$);
    $\mathcal{E}_{\mathcal{M}} \leftarrow \mathcal{E}_{\mathcal{M}} \cup \{(i_{P_R}, i_D, i_R)\}$;
  end
end
$\mathcal{E} \leftarrow \mathcal{E}_{\mathcal{O}} \cup \mathcal{E}_{\mathcal{M}}$;

Algorithme 12: Création de la configuration de messages

Génération de code pour la création d'une configuration de messages

pp-createMsgConfiguration($\mathcal{E}_{\mathcal{M}}$)
$\rightsquigarrow$ *eq Config-Msg* = $Msg_1(i_{D_1}, i_{R_1})$ ... $Msg_n(i_{D_n}, i_{R_n})$.
avec $\mathcal{E}_{\mathcal{M}}$ = {(Msg_1,i_{D_1}, i_{R_1}), ...,(Msg_1,i_{D_1}, i_{R_1})} }

5) La fonction $\mathtt{Trad}_R$ *: Création des règles de réécriture et déclaration des types*

La fonction $\mathtt{Trad}_R$ a pour objectif de créer l'ensemble des règles de comportement à partir des règles utilisateur modélisées dans l'ontologie par les instances du concept *Règle-utilisateur*. Ces règles sont formées de prédicats binaires représentant des propriétés de l'ontologie. Chaque prédicat peut avoir comme argument des individus, des littéraux ou des variables. Les objets existants ont été déclarés (dans Σ_0) et créés (dans $\mathcal{E}$) à l'étape précédente (cf. section 4.7.3). Les variables et littéraux restent néanmoins à déclarer. La fonction $\mathtt{Trad}_R$ prend en entrée l'ensemble des instances $\mathbb{I}_{RU}$ du concept *Règle-utilisateur*, l'ensemble des axiomes terminologiques, l'ensemble de typage Σ_0 résultat de la fonction de traduction $\mathtt{Trad}_{E_O}$ et la fonction $\mathfrak{I}^{\mathbb{P}}$. Le processus de création des règles de réécriture est décrit par l'Algorithme 13. L'algorithme traite l'ensemble des valeurs des propriétés *Antécédent* et *Conséquent* de chaque instance du concept *Règle-Utilisateur*. Ces valeurs de propriétés correspondent à des prédicats. Les fonctions *getPredicateName*, *getPredicateDomainValue* et *getPredicateRangeValue* sont appelées afin d'obtenir le nom du prédicat (la propriété à laquelle il réfère) et ses valeurs de domaine et d'image qui peuvent chacune correspondre à un individu, une variable ou un littéral. Les prédicats d'une règle utilisateur peuvent représenter des relations ou des attributs définis dans l'ontologie. On distingue deux types d'attributs : dynamique et statique (cf. section 3.1.1.1). La figure ci-après donne un exemple de transformation d'une règle utilisateur (en (1)) en une règle de réécriture (en (2)). Les arguments du domaine des prédicats sont transformés en objets : i_x devient O_x et i_y devient O_y. Les prédicats représentant des relations sont transformés en messages : $R_1(i_x,i_y)$ devient $Msg_1(O_x,O_y)$. Les prédicats représentant des attributs A_i sont transformés en attributs d'objets A_i. Les attributs dynamiques tel que A_1 peuvent apparaître en conséquent d'une règle utilisateur, la valeur d'attribut de l'objet possédant l'attribut dynamique (O_x dans l'exemple) doit alors être actualisée : $A_1 : O_z$ devient $A_1 : O_w$.

$$\textbf{1)}\ A_1(i_x,i_z) \wedge A_2(i_x,i_t) \wedge A_3(i_y,i_t) \wedge R_1(i_x,i_y) \rightarrow A_1(i_x,i_w)$$
$$\textbf{2)}\ <O_x : C_x|\ A_1 : O_z,\ A_2 : O_t> <O_y : C_y|A_3 : O_t>\ Msg_1(O_x,O_y)$$
$$\rightarrow <O_x : C_x\ |\ A_1 : O_w, A_2 : O_t> <O_y : C_y|A_3 : O_t>$$

Transformation d'une règle utilisateur en règle de réécriture.

Au fur et mesure du parcours des prédicats d'une règle utilisateurs les objets sont créés ou leurs valeurs d'attributs actualisées lorsqu'ils existent déjà. La fonction *updateObjects* permet cela. Elle prend en entrée un ensemble d'objets et un objet à y ajouter (cf . algorithme 13).

L'ensemble des individus ayant été déclaré durant l'exécution de la fonction de traduction $Trad_{E_O}$, seul le cas où les valeurs de domaine ou d'image des prédicats correspondent à des variables ou des littéraux entraîne de nouvelles déclarations dans l'espace de typage Σ. Enfin une règle de réécriture Maude est constituée à partir des valeurs de prédicats de chaque instance de *Règle-utilisateur*.

Génération de code pour la déclaration de variables

pp-declareVariables(Σ)
$$\rightsquigarrow \textit{var}\ v_1 : t_1\ .$$
$$\ldots$$
$$\textit{var}\ v_n : t_n\ .$$
$$\textit{avec}\ \Sigma = \{\{(v_1,t_1), \ldots, (v_n,t_n)\}, \Sigma_0\}$$

Génération de code pour la déclaration de règles de réécriture

pp-createRules($\mathcal{R}$)
$$\rightsquigarrow \textit{rl}\ : O_i \ldots O_n\ Msg_i \ldots Msg_u \Rightarrow O'_i \ldots O'_n\ Msg_l \ldots Msg_x\ .$$
$$\ldots$$
$$\textit{rl}\ : O_j \ldots O_m\ Msg_j \ldots Msg_t \Rightarrow O'_j \ldots O'_m\ Msg_k \ldots Msg_z\ .$$
$$\textit{avec}\ \mathcal{R} = \{\ (\{(O_i, \ldots,O_n),(Msg_i \ldots Msg_u)\},\{(O'_i \ldots O'_n),(Msg_l \ldots Msg_x)\}), \ldots, (\{(O_j, \ldots,O_m),(Msg_j \ldots Msg_t)\},\{(O'_j \ldots O'_m),(Msg_k \ldots Msg_z)\})\ \}$$

4.8 CONCLUSION

Dans ce chapitre, nous avons présenté une méthode de transformation de spécifications OWL en spécifications Maude. Après avoir défini les éléments de correspondance entre OWL et Maude, nous avons décrit le passage du modèle ontologique au modèle Maude. Notre méthode de transformation repose sur l'exploitation de fonctions de manipulation d'éléments ontologiques. Ces fonctions permettent d'accéder directement aux connaissances nécessaires à la définition des spécifications Maude, notre objectif n'étant pas de traduire l'ensemble de l'ontologie mais seulement les éléments permettant d'étendre les possibilités de vérification offertes par OWL. Nous avons montré que

Input : $\mathbb{I}_{RU}$,$\mathfrak{I}^{\mathbb{P}}$, $\mathbb{P}_R$, $\mathbb{A}$, Σ_0
Output : $\mathcal{R}$,Σ
$\mathcal{R} \leftarrow \emptyset$; $\Sigma \leftarrow \Sigma_0$; *Objs-Antécédent* $\leftarrow \emptyset$;
Msg-Antécédent $\leftarrow \emptyset$; *Msg-Conséquent* $\leftarrow \emptyset$;
Class-Attributes-Values $\leftarrow \emptyset$;
for *each* i_{RU} *in* $\mathbb{I}_{RU}$ **do**
  A-predicates $\leftarrow$ *getRangeValue*(i_{RU},*Antécédent*,$\mathfrak{I}^{\mathbb{P}}$) ;
  C-predicates $\leftarrow$ *getRangeValue*(i_{RU},*Conséquent*,$\mathfrak{I}^{\mathbb{P}}$) ;
  for *each a-predicate in A-predicates* **do**
    $p \leftarrow$ *getPredicateName*(*a-predicate*) ;
    $v_D \leftarrow$ *getPredicateDomainValue*(*a-predicate*) ;
    $v_R \leftarrow$ *getPredicateRangeValue*(*a-predicate*) ;
    $t_D \leftarrow$ *getDomain*(p,$\mathbb{A}$) ;
    $t_R \leftarrow$ *getRange*(p,$\mathbb{A}$) ;
    if isVariableOrLitteral(v_D) **then**
      $\Sigma \leftarrow \Sigma \cup \{(v_D,t_D)\}$;
    if isVariableOrLitteral(v_R) **then**
      $\Sigma \leftarrow \Sigma \cup \{(v_R,t_R)\}$;
    if $p \in P_R$ **then** //p est une relation
    Msg-Antécédent $\leftarrow$ *Msg-Antécédent* $\cup \{(p, v_D, v_R)\}$;
    Objs-Antécédent $\leftarrow$ *updateObjects*(*Objs-Antécédent*,$\{(v_D,t_D,\emptyset,\emptyset)\}$);
    Objs-Antécédent $\leftarrow$ *updateObjects*(*Objs-Antécédent*,$\{(v_R,t_R,\emptyset,\emptyset)\}$);
    else //p est un attribut
    Objs-Antécédent $\leftarrow$ *updateObjects*(*Objs-Antécédent*,$\{(v_D,t_D,p,v_R)\}$)
  Objs-Conséquent $\leftarrow$ *Objs-Antécédent*;
  for *each c-predicate in C-predicates* **do**
    $p \leftarrow$ *getPredicateName*(*c-predicate*) ;
    $v_D \leftarrow$ *getPredicateDomainValue*(*c-predicate*) ;
    $v_R \leftarrow$ *getPredicateRangeValue*(*c-predicate*) ;
    $t_D \leftarrow$ *getDomain*(p,$\mathbb{A}$) ;
    $t_R \leftarrow$ *getRange*(p,$\mathbb{A}$) ;
    if *isDynamic*(p,$\mathbb{A}$) **then** //p est un attribut dynamique
    Objs-Conséquent $\leftarrow$ *updateObjects*(*Objs-Conséquent*,$\{(v_D,t_D,p,v_R)\}$)
    else
      if $p \in P_R$ **then** //p est une relation
      Msg-Conséquent $\leftarrow$ *Msg-Conséquent* $\cup \{(p, v_D, v_R)\}$;
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{$(*Objs-Antécédent*,*Objs-Conséquent*,Msg-Antécédent,Msg-Conséquent)$\}$;

Algorithme 13: Création des règles de réécriture et déclaration des types

ce passage permet effectivement de compléter les vérifications permises par OWL. La correction de la traduction entre le modèle ontologique et le modèle Maude a été vérifiée de manière empirique, sa vérification formelle est discutée dans nos perspectives.

5

APPLICATION ET RÉSULTATS

Sommaire

« L'important, le principal est de savoir ce qu'il faut observer. »

Edgar Allan Poe

INTRODUCTION

Dans ce chapitre, nous introduisons le projet *ENVIE VERTE* [1] [2] dans lequel s'inscrit ce travail. Nous poursuivons sur la description du domaine des environnements intelligents qui correspond au domaine d'application. Nous détaillons ensuite l'ontologie générique proposée pour modéliser ce domaine, la termino-ontologie et les résultats d'acquisition de termes. Puis, nous décrivons le corpus dont nous disposons pour évaluer notre système. Enfin nous détaillons les résultats pour les différentes étapes du processus de formalisation.

5.1 DESCRIPTION DU PROJET ENVIE VERTE

ENVIE VERTE est un projet interdisciplinaire porté par le *LIMSI-CNRS* et regroupant le *LIMSI-CNRS*, le *CEDRIC-CNAM* et l'institut *Gaspard Monge*. Ce projet rassemble des travaux relevant de plusieurs disciplines telles que le traitement automatique des langues, la représentation des connaissances, les méthodes formelles et le réseaux de capteurs. L'objectif visé consiste à permettre à un utilisateur de configurer le comportement de son environnement intelligent, en décrivant ses besoins en langage naturel. Dans ce contexte, les spécifications sont rédigées par un utilisateur non expérimenté à l'art de spécifier des exigences. Il est donc important de lui faire un retour précis lorsque celles-ci doivent être corrigées. La finalité du projet est de procéder au déploiement du système modélisé lorsque son modèle est cohérent et conforme aux exigences. Le déploiement de l'environnement intelligent, ou plus exactement du réseau de capteurs qui décrit son comportement, s'inscrit dans le cadre du projet ENVIE VERTE. Néanmoins, il est hors du cadre de notre travail de thèse.

5.2 DESCRIPTION DU DOMAINE DES ENVIRONNEMENTS INTELLIGENTS

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.

1. Financé par *DIGITEO*, projet DIM LSC 2010.
2. http ://envieverte.limsi.fr/

Un *capteur*, ou *détecteur*, est un appareil de détection ou de mesure, sensible à différents types de phénomènes. Il est destiné à détecter l'occurrence d'un phénomène ou à quantifier sa grandeur physique. Un capteur transforme l'information captée en signal, souvent électrique, qui indique la manifestation ou la grandeur physique d'un phénomène. Le signal renvoyé est utilisable pour l'application d'une action particulière.

Un *actionneur* est un dispositif qui entre en activité lorsqu'il reçoit un ordre ou un signal particulier émanant d'un capteur. Il convertit l'information en une action particulière en fonction du signal perçu qui lui est fourni.

Les *processus de contrôle* définissent les règles de communication entre les différents dispositifs de l'environnement intelligent.

Dans un environnement intelligent, on peut distinguer deux niveaux de description : une description matérielle qui correspond aux différents équipements, leur nombre, leur type, leur localisation etc. et une description logicielle qui correspond aux règles de comportement auxquelles est soumis l'environnement intelligent. C'est le traitement de cette seconde partie qui est abordée dans l'approche que nous proposons.

La profusion et la diversité des capteurs laisse imaginer une multitude de configurations telle que l'environnement intelligent puisse s'adapter aux besoins d'un utilisateur. Néanmoins, les environnements intelligents sont soumis au comportement général décrit ci-dessous :

- Un capteur détecte l'occurrence d'un phénomène ou mesure sa grandeur dans une zone restreinte ;
- Un capteur détecte ou mesure un type de phénomène s'il est localisé dans sa zone de capture ;
- Un actionneur est connecté à un appareil (processus physique) de l'environnement qu'il peut actionner ;
- La détection d'un phénomène peut conduire à l'activation d'un ou plusieurs actionneurs et actionner une ou plusieurs actions (*allumer*, *ouvrir*, *augmenter* ...) sur les processus physiques qu'ils contrôlent ;
- Un actionneur, pour être activé par un capteur, doit gérer le type de phénomène perçu par le capteur et être localisé dans sa zone de contrôle.

5.3 DÉVELOPPEMENT DE L'ONTOLOGIE 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*[3]. 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 un environnement d'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*[4] 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],Russomanno *et al.* [2005], Wang et Turner [2009], Khattak *et al.* [2011] et Albreshne *et al.* [2013] modélisent les réseaux de capteurs à l'aide d'ontologies afin de permettre une inférence ou une interrogation sur les diverses connaissances représentées. Dans ce cas, la définition des ontologies repose en général sur une hiérarchisation profonde des concepts.

Dans l'approche de modélisation que nous proposons, l'ontologie doit privilégier trois points : 1) une distinction claire entre les composants du système et les types permettant de les caractériser ; 2) une définition précise des domaine et image de chaque propriété et une définition des concepts par leurs propriétés définissantes ; 3) la représentation des règles de comportement générique propre au domaine. À notre connaissance, ce type d'ontologie qui modéliserait le comportement d'un environnement intelligent n'existe pas. Aussi, la modélisation de l'ontologie est effectuée à partir de l'étude des connaissances du domaine pour identifier les concepts clés ainsi que leurs propriétés définissantes. 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[5].

De l'étude du domaine des environnements intelligents et plus particulièrement des réseaux de capteurs, il ressort que l'ensemble des capteurs et actionneurs sont principalement caractérisés par le type

3. Sensor Model Language Encoding Standard
4. http ://www.opengeospatial.org/
5. http ://www.w3.org/TR/owl2-overview/

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. Ainsi, les individus des concepts *Sensor*, *Actuator* et *Phenomenon* peuvent être caractérisés par un type de phénomène et une localisation. De manière analogue, une localisation est caractérisée par un type (chambre, cuisine, salle de bains...) et peut elle-même être localisée dans une ou plusieurs localisations (maison, rez-de-chaussée ...).

5.3.1 *Modélisation de l'ontologie du comportement d'un environnement intelligent*

La figure 21 décrit l'ontologie du comportement d'un environnement intelligent. L'ontologie est divisée en deux parties : une partie haute en noir et blanc qui définit les concepts de haut niveau qui sont génériques au comportement d'un système à base de règles et une partie basse en couleur qui correspond au domaine d'application, en l'occurrence celui des environnements intelligents. La partie spécifique aux environnements intelligents comporte quatorze concepts (cf. tableau 2) : sept sous-concepts de *Composant*, illustrés en jaune foncé dans la figure et sept sous-concepts de *Type* illustrés en jaune clair. Dans l'ontologie, 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.

Sous-concept *Composant*	**Sous-concept *Type***
Actuator	State
Sensor	Value
Detecting-sensor	Event
Measuring-sensor	Measurable
Location	Location-Type
Physical-process	Physical-process-Type
Phenomenon	Phenomenon-Type

Tableau 2: Concepts de l'ontologie

Notre modélisation repose sur la définition de propriétés caractérisant les concepts et leurs individus. Ainsi, les individus de l'ontologie se distinguent en fonction de leurs valeurs de propriétés définissantes plutôt que de leur emplacement dans la hiérarchie de concept. Cha-

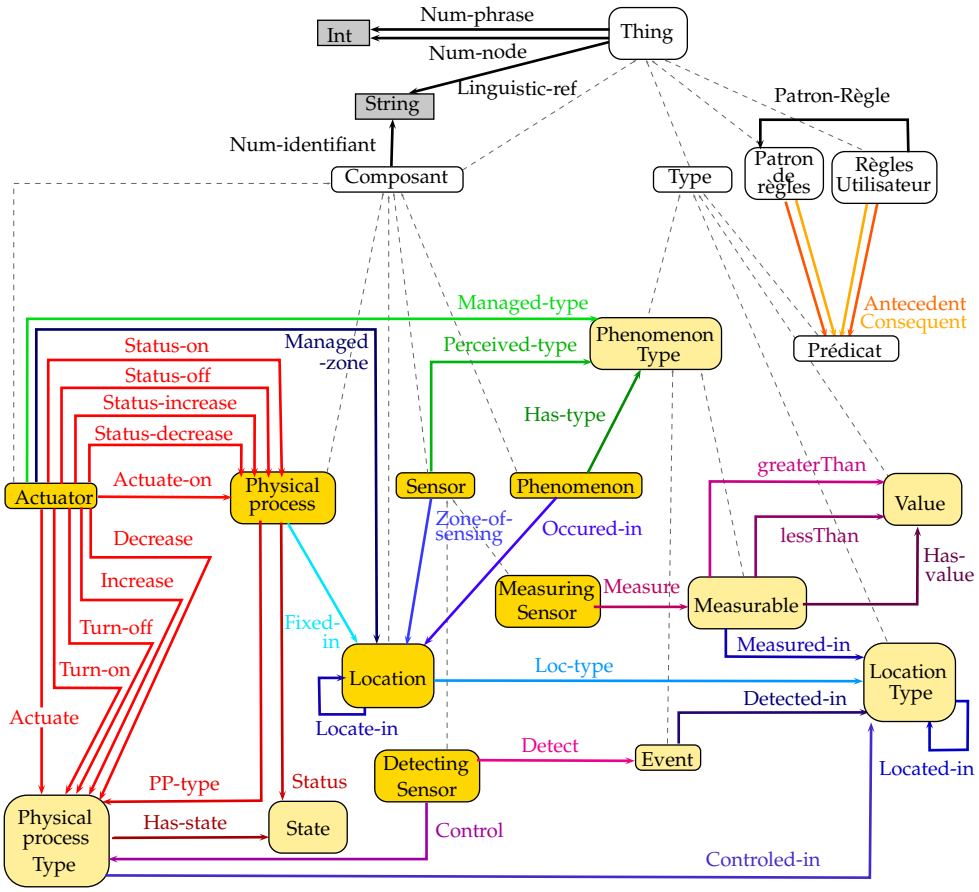


FIGURE 21: Ontologie du comportement d'un environnement intelligent

cune des propriétés définies est caractérisée par un seul concept de domaine et un seul concept d'image, ce qui permet de distinguer des propriétés dont la sémantique est assez proche tels que *Fixed-in*, *Zone-of-sensing*, *Occured-in* ou *Controled-in*. Ce choix permet de restreindre la sémantique véhiculée par chaque propriété et ainsi limiter les ambiguïtés lors de l'analyse des spécifications. Les propriétés caractérisant la possession d'une localisation ou d'un type de phénomène se déclinent donc au sein de l'ontologie en fonction des concepts qu'elles lient.

5.3.2 *Caractéristiques des propriétés de l'ontologie*

OWL permet d'associer aux propriétés différentes caractéristiques. Certaines propriétés ont ainsi pu être définies comme *fonctionnelles* ou *transitives*. Les propriétés fonctionnelles portent sur les individus ne pouvant posséder qu'une seule valeur d'image. Elles permettent d'inférer l'égalité entre des individus, lorsque ceux-ci sont liés à une même valeur d'image au travers d'une même propriété fonctionnelle. Elles permettent en plus de contraindre les valeurs possibles des pro-

priétés. Les propriétés transitives permettent d'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.

Dans notre modèle, la propriété *Locate-in* qui lie deux individus du concept *Location* est transitive. Cela permet l'inférence suivante : Si l est localisé dans l_1 et l_1 est localisé dans l_2 alors l est localisé dans l_2.

$$\textit{Locate-in}(l, l_1) \textit{ et } \textit{Locate-in}(l_1, l_2) \textit{ alors } \textit{Locate-in}(l, l_2) \quad \textbf{(1)}$$

Quant aux propriétés *Fixed-in*, *Zone-of-sensing* et *Occurred-in*, leur transitivité est inférée à l'aide de règles SWRL définies sur la propriété *Locate-in* comme l'illustre la règle SWRL (2). Cette règle indique que lorsqu'un appareil, ou processus physique ?p est installé dans une localisation ?l et que ?l est lui-même localisé dans $?l_2$ alors ?p est aussi installé dans $?l_2$. La règle SWRL (3) illustre un exemple d'instanciation de la règle (2).

$$\textit{Fixed-in}(?p, ?l) \wedge \textit{Locate-in}(?l, ?l_2) \rightarrow \textit{Fixed-in}(?p, ?l_2) \quad \textbf{(2)}$$

$$\textit{Fixed-in(radiator-kitchen, kitchen)} \wedge \textit{Locate-in(kitchen, ground floor)} \rightarrow \textit{Fixed-in(radiator-kitchen, ground floor)} \quad \textbf{(3)}$$

Comme nous l'avons abordé dans les chapitres 2 et 3, les entités qui correspondent aux individus du concept ou des sous-concepts de *Composant* ne sont généralement pas directement reconnaissables à partir des textes. Seules les instances de propriétés auxquelles elles participent le sont. Dans les spécifications, les instances de propriétés à rechercher sont donc celles auxquelles participent les individus du concept *Type*. Par exemple, pour être reconnu, un individu du concept *Phenomenon* doit avoir ses deux valeurs de *type* et de localisation définies, respectivement pour ses propriétés *Has-type* et *Occurred-in*.

5.3.3 *Définition de règles d'inférences*

Dans certaines formulations, un terme peut connoter ou désigner une propriété qu'un individu doit posséder [Chomsky et Dubois-Charlier, 1969]. La valeur de propriété de l'individu est alors caractérisée directement par le terme qui le dénote 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'une règle utilisateur 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 où le type de phénomène reconnu est *temperature*.

$$a)\ \ldots\ \textit{Measured-in}(t,l)\ \textit{Has-type}(?p,t)\ \ldots \Rightarrow \ldots$$

$$b)\ \ldots\ \textit{Measured-in}(\textbf{\textit{temperature}},\textit{kitchen})\ \ \textit{Has-type}(?\mathrm{p},\textbf{\textit{temperature}})\ \ldots \Rightarrow \ldots$$

Enfin, un ensemble de règles SWRL permet de déduire les individus identiques du concept *Composant*, ce qui permet de lier les individus issus des textes aux individus pré-existant dans l'ontologie. Ces règles exploitent pour cela les propriétés définissantes de chaque concept, comme l'illustre l'exemple de règle ci-dessous. Cette règle 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.

$$\textit{Perceived-type}(?s_1, ?t) \wedge \textit{Perceived-type}(?s_2, ?t) \wedge \textit{Zone-of-sensing}(?s_1, ?l) \wedge \textit{Zone-of-sensing}(?s_2, ?l) \rightarrow \textit{sameAs}(?s_1, ?s_2)$$

L'ontologie ainsi définie offre un cadre d'interprétation formelle des spécifications d'exigences du comportement d'un environnement intelligent. Dans le contexte du projet ENVIE VERTE, ces exigences ont été recueillies au travers de la plate-forme d'acquisition de spécifications d'exigences décrite dans la section suivante.

5.4 PLATE-FORME D'ACQUISITION DE SPÉCIFICATIONS D'EXIGENCES

L'objectif du projet ENVIE VERTE est de permettre à un utilisateur de configurer son propre environnement intelligent à l'aide de spécifications écrites en langage naturel. 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 [6]. Une fois l'utilisateur connecté, une interface de rédaction des spécifications lui est proposée (cf. figure 22). 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. L'utilisateur peut alors spécifier ses besoins en langage naturel et les enregistrer lorsque ceux-ci lui conviennent. Les spécifications sont enregistrées de manière à garantir à l'utilisateur l'accès à ses précédentes spécifications lors d'une connexion suivante, pour les étendre ou les

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

modifier. Dans la section *Perspectives*, l'évolution de cette plate-forme en une interface d'aide à la rédaction de spécifications est abordée.

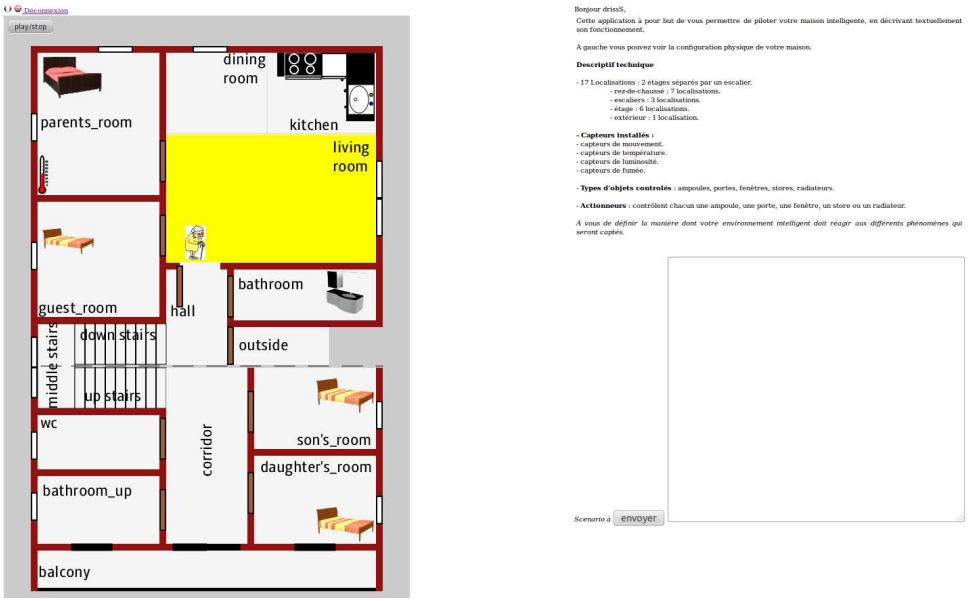


FIGURE 22: Interface de rédaction de spécifications d'exigences

5.5 PRÉTRAITEMENTS

Les prétraitements que nous présentons dans cette section ne sont pas spécifiques au domaine d'application. Ils relèvent de l'implémentation du système de formalisation des spécifications d'exigences. Les prétraitements précèdent l'acquisition de règles d'extraction et s'effectuent sur les résultats de l'analyse syntaxique des textes. L'analyseur syntaxique utilisé est le *Stanford Parser*, lequel produit des arbres de dépendances syntaxiques à partir des phrases du texte. Les prétraitements consistent en trois traitements successifs illustrés en figures 23 et décrits ci-dessous.

5.5.1 *Lemmatisation des mots de l'arbre de dépendances*

Dans les arbres de dépendances générés par le *Stanford Parser*, nous ramenons chaque mot contenu dans un nœud à sa forme lemmatisée, appelée *lemme* et fournie lors de l'analyse syntaxique. Un lemme correspond à la forme canonique d'un mot c'est-à-dire

1. le singulier, masculin pour les noms et les adjectifs ;
2. l'infinitif pour les verbes.

La lemmatisation présente deux avantages : 1) réduire la taille de la terminologie nécessaire à la définition du domaine car elle ne contient

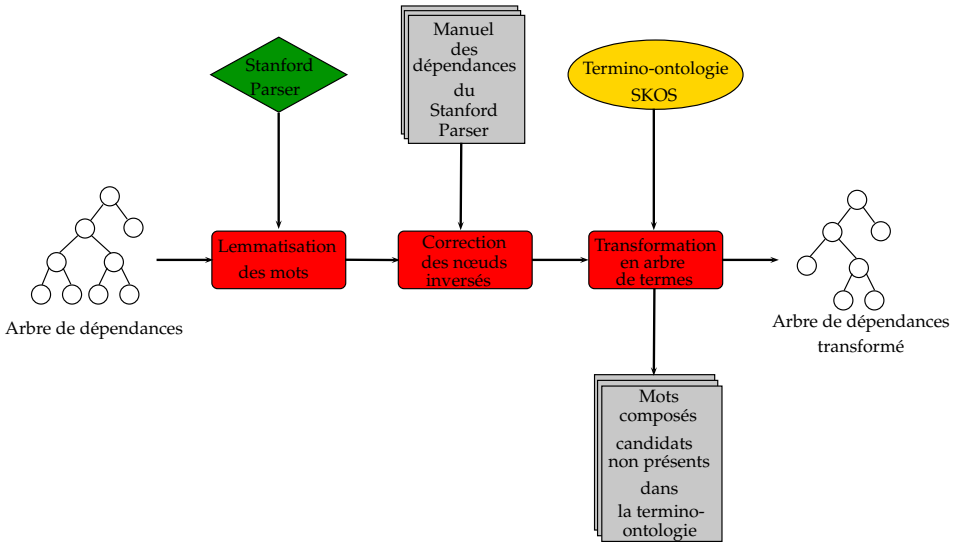


FIGURE 23: Chaîne de traitement des arbres de dépendances syntaxiques

alors que les formes canoniques des mots ; 2) réduire de manière significative le nombre de triplets à rechercher dans les textes.

5.5.2 *Correction des dépendances syntaxiques*

Notre approche d'acquisition de règles d'extraction est fondée sur l'exploitation des caractéristiques récurrentes des dépendances syntaxiques formant des chemins entre triplets de termes. Une dépendance syntaxique constitue un lien orienté, aussi l'ordre des nœuds qu'elle lie est une donnée importante. Néanmoins, lors de la génération des arbres de dépendances, il peut arriver que le Stanford Parser inverse l'ordre des nœuds contenus dans certaines dépendances syntaxiques. Ce type d'erreur peut mener à la construction de chemins incorrects. Afin de pallier ce désagrément, chaque arbre de dépendances est traité de manière à rechercher d'éventuelles inversions des nœuds d'une dépendance pour la corriger. Ce traitement consiste en une vérification systématique des catégories syntaxiques des nœuds de chaque dépendance syntaxique produite par le *Stanford Parser*, guidée par le manuel des types de dépendances [De Marneffe et Manning, 2008] qui définit pour chacune les catégories syntaxiques associées à ses nœuds *gouverneur* et *dépendant*. Par exemple, dans ce manuel, la dépendance *prep* est définie comme représentant une préposition qui sert à modifier le sens d'un verbe (*VB*), d'un adjectif (*JJ*), d'un nom (*NN*) ou même d'une autre préposition (*IN*). La préposition notée *IN* correspond alors systématiquement au nœud dépendant i.e. le deuxième argument de la dépendance *prep*. L'exemple ci-dessous

illustre une transformation effectuée sur une dépendance de type *prep* dont les nœuds ont été inversés.

*prep(on-**IN**-8,switch-**VB**-7)* → *prep(switch-**VB**-7,on-**IN**-8)*

Suite à l'analyse des sorties du *Stanford Parser*, près de *trois cents milles* dépendances ont pu être corrigées sur un total d'environ *trois millions*.

5.5.3 *Transformation des mots en termes*

Les analyseurs syntaxiques segmentent chaque phrase en mots afin de créer des structures arborescentes dont les feuilles contiennent ces mots. Dans notre contexte, l'unité lexicale de base considérée est le terme qui peut être composé d'un ou plusieurs mots. Les termes, lorsqu'ils sont composés de plusieurs mots, ne sont alors pas reconnus par le *Stanford Parser*. Afin d'exploiter les arbres de dépendances sur des termes, le contenu de leurs nœuds doit être transformé en termes. Il s'agit alors de rechercher dans les arbres de dépendances les nœuds dont les mots peuvent former un terme composé. Cela consiste à modifier le contenu des nœuds et parfois même à supprimer ou ajouter des dépendances et donc à réécrire des sous-arbres.

Afin de reconnaître les contextes dans lesquels apparaissent les termes composés, la méthode de recherche de chemins entre triplets de termes a été adaptée à la reconnaissance de chemins liant les mots d'un même terme. À partir des chemins acquis, des règles de transformation ont été manuellement écrites. Ces règles ont pour *antécédent* un chemin syntaxique liant les mots d'un terme composé et en conséquent un chemin syntaxique actualisé i.e. contenant le terme et non plus les mots qui le composent. Les règles de transformation écrites sont appliquées aux résultats de l'analyse syntaxique du corpus. Chaque application a deux conséquences possibles : 1) le terme composé reconnu existe dans la termino-ontologie, les nœuds qui le contiennent doivent alors être modifiés ; 2) le terme composé reconnu n'existe pas dans la termino-ontologie, il est alors stocké dans un fichier comme *terme candidat* de la termino-ontologie *SKOS*.

En tout, 11 règles de transformation ont été constituées. Ces règles peuvent être scindées en deux grandes catégories : les règles qui opèrent une transformation sur le contenu des nœuds des dépendances, comme par exemple (a) et (b), et les règles qui en plus de la transformation du contenu d'un nœud, opèrent également une transformation sur les dépendances syntaxiques, comme par exemple (c). Cette seconde catégorie permet en plus de corriger certaines des dépendances syntaxiques incorrectes.

La règle de transformation (a) s'applique lorsqu'une dépendance de type *adjectif modificateur* (*amod*) a pour premier argument un nom (*NN*) et pour second argument un adjectif (*JJ*), elle crée alors un *terme candidat* à partir de la combinaison *adjectif nom*.

a) *amod(bulb-**NN**-3, light-**JJ**-2) → amod(light bulb-**NN**-3, light-**JJ**-2)*

La règle de transformation (b) s'applique lorsqu'une dépendance de type *nom modificateur* (*nn*) a pour premier argument un nom (*NN*) et pour second argument un nom (*NN*). Elle crée alors un *terme candidat* à partir de la combinaison *nom (second argument) nom (premier argument)*.

b) *nn(room-**NN**-3, dining-**NN**-2) → nn(dining room-**NN**-3, dining-**NN**-2)*

La règle (c) permet de corriger une erreur assez fréquente du *Stanford Parser* qui considère la particule (ici *on*) d'un verbe (ici *switch on*) comme une préposition (IN) ce qui mène à la création de deux dépendances incorrectes. Dans ce cas, la correction consiste à rattacher le verbe à sa particule et de créer une nouvelle dépendance de type *objet direct* à partir de ce nouveau terme et de supprimer les deux dépendances incorrectes.

c) *prep(switch-**VB**-7,on-**IN**-8) pobj(light-**NN**-11,on-**IN**-8) → dobj(switch on-VB-7, light-NN-8)*

5.6 ACQUISITION DES RÈGLES ET DE LA TERMINOLOGIE DU DOMAINE

5.6.1 *Description des ressources*

En l'absence de corpus suffisamment grand portant sur la spécification d'exigences dans le domaine des environnements intelligents, nous avons constitué un corpus d'apprentissage à partir de livres électroniques (e-books) de domaines et de styles littéraires variés, issus de la *Bibliothèque numérique Anacleto*[7]. Le corpus constitué comporte plus de 4 millions de mots en anglais. La diversité de ce corpus permet d'acquérir des règles pour des styles d'écriture différents. Les propriétés à acquérir étant trans-domaines, le recours à un tel corpus pour la reconnaissance des règles d'extraction est fondé. Cependant sa généralité constitue un handicap pour l'extraction de la terminologie liée aux concepts qui sont spécifiques au domaine des environnements intelligents.

7. (http ://www.gutenberg.us/)

La termino-ontologie amorce est construite manuellement, à partir des termes préférés de chaque concept ou propriété et de quelques synonymes. Elle contient 278 termes dont 32 préférés et 246 alternatifs (cf. tableau 3). Les termes préférés portent le label *prefLabel*. Ils sont tous uniques et réfèrent chacun à un individu d'un sous-concept de *Type* existant dans l'ontologie. Les termes alternatifs correspondent aux différentes formulations pouvant dénoter les individus dans la langue. Sur les 246 termes alternatifs 217 sont différents dont 109 termes hors valeurs numériques. Parmi ces 109 termes, on compte aussi des termes et leurs déclinaisons tels que *living room* et *living-room* ou *turn down* et *turn-down*.

Label	**Nb termes**
altLabel	246 (217 différents - 109 hors nombres)
prefLabel	34
implicit	7
partial	5

Tableau 3: Termes de la termino-ontologie amorce

5.6.2 *Acquisition des termes*

L'approche d'acquisition de règles d'extraction d'instances proposée s'appuie sur la reconnaissance des termes issus de la termino-ontologie dans les textes. La définition de l'ensemble des termes du domaine de manière exhaustive n'est pas envisageable. Aussi, nous partons d'une termino-ontologie amorce contenant un petit ensemble de termes dénotant chaque instance de concept ou de propriété. Cette termino-ontologie est le point de départ du processus d'acquisition des règles et des termes par amorçage (cf. figure 24). Ce processus prend en entrée les résultats d'analyse syntaxique du corpus d'apprentissage, à partir desquels sont acquis les chemins syntaxico-sémantiques entre triplets de termes. Chaque chemin, engendre la création de trois fonctions *Java* pour l'extraction de termes particuliers à chacun des trois ensembles sémantiques impliqués. Les chemins entre triplets partiels n'engendrent que deux fonctions *Java*. Chacune des fonctions d'extraction de termes est ensuite appliquée aux résultats d'analyse syntaxique du corpus d'apprentissage pour en extraire de nouveaux termes et de nouvelles règles qui serviront à étendre la termino-ontologie. Les itérations se succèdent ainsi jusqu'à ce que plus aucun nouveau terme ou nouvelle règle ne soit acquis.

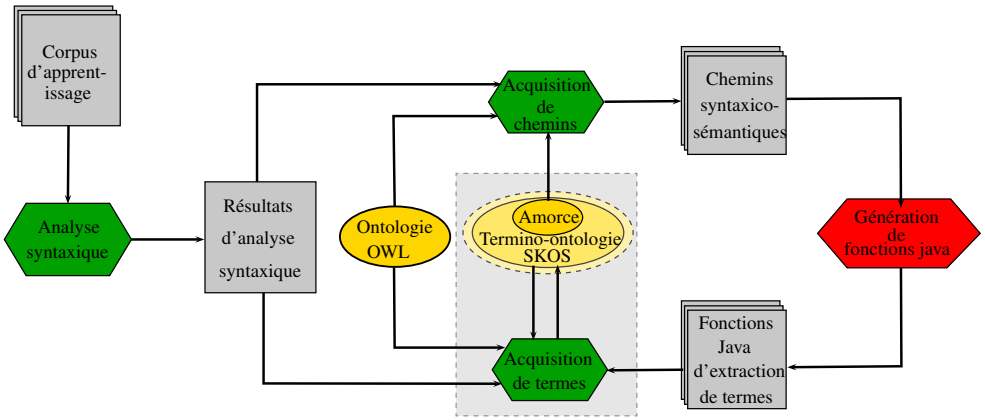


FIGURE 24: Acquisition de termes par amorçage

5.6.3 *Évaluation de l'acquisition de termes*

Nous avons empiriquement fixé le nombre maximum de dépendances d'un chemin syntaxique à 4 et le nombre de chemins les plus fréquents à retenir aux 4 premiers avec leurs déclinaisons, i.e. les chemins dont les dépendances syntaxiques diffèrent uniquement par les catégories syntaxiques. 495 règles d'extraction de termes (cf. section 3.3.5) ont été générées. Chaque règle acquise vise l'extraction d'un type de terme i.e. qui dénote une classe sémantique particulière (concept ou propriété). Ces règles d'extraction ont été appliquées sur le corpus d'apprentissage comme décrit en section 3.3.5.

La pertinence des termes extraits est calculée en fonction du nombre de fois qu'ils sont extraits par une même règle et le nombre de règles du même type l'ayant extrait. Les termes sont alors classés selon la mesure de pertinence (cf. section 3.3.5). Nous avons fixé à 50 le nombre de termes les plus fréquents retournés. La sélection parmi ces termes est effectuée de façon manuelle pour deux raisons : 1) le corpus de textes est trop générique pour contenir suffisamment d'occurrences de termes propres au domaine des environnements intelligent pour permettre une sélection automatique pertinente ; 2) certains termes partagent les mêmes contextes de formulation sans pour autant dénoter les mêmes classes sémantiques. Par exemple, les termes *leave* et *enter* qui désignent chacun un mouvement, i.e. un phénomène sensiblement différent (*movement in* et *movement out*). De même les deux termes *open* et *close* désignent chacun une action, qui néanmoins s'opposent. Dans ces cas là, le choix de la classe sémantique appropriée dépend essentiellement du terme dont il est question. L'intervention humaine s'impose alors pour affecter les termes aux bonnes classes sémantiques.

Au total, trois itérations ont été appliquées sur le corpus d'apprentissage. À chaque itération la termino-ontologie a été augmentée par

les termes sélectionnés. Ces termes sont essentiellement le résultats des règles d'extraction de termes et pour une plus faible proportion des termes candidats composés durant la phase de transformation des arbres de dépendances en arbres de termes (cf. section 5.5.3) dont le nombre est représenté entre parenthèses dans la colonne It_3 du tableau 4. Les tableaux 4 et 5 montrent respectivement les résultats de l'acquisition des termes référant aux individus des sous-concepts du concept *Type* et des termes référant aux propriétés de type attribut. La colonne *Amorce* décrit le nombre de termes amorces. Les colonnes It_i décrivent le nombre de termes acquis à l'itération i. Parmi l'ensemble des termes de la termino-ontologie résultat on compte 10% de termes ambigus, c'est-à-dire qui appartiennent à plus d'une classe sémantique.

Concept	**Amorce**	It_1	It_2	It_3	**Total**
Event	26	3	7	2 + (9)	21
Measurable	4	2	1	1	4
Value	108	5	1	2 + (7)	15
Location-Type	22	1	9	1	11
Physical-process-Type	14	5	9	4 + (9)	27
State	4	3	3	2	8

Tableau 4: Résultat de l'extraction des termes dénotant un sous-concept de *Type*

5.7 ÉVALUATION DE L'EXTRACTION D'INSTANCES DE PROPRIÉTÉS

5.7.1 *Description des ressources*

À partir de la plate-forme d'acquisition de spécifications d'exigences (cf. section 5.4), plus d'une centaine de phrases en anglais contenant 2171 mots ont été collectées. Celles-ci sont rédigées par une vingtaine de personnes majoritairement francophones. Afin de disposer de cas d'incohérences, ces vingt spécifications sont considérées comme une seule spécification utilisateur.

L'évaluation d'un système doit se faire par rapport à une référence. Afin de construire cette référence, les spécifications collectées ont été annotées selon le protocole décrit dans la section suivante.

Propriété	**Amorce**	It_1	It_2	It_3	**Total**
Turn-on	12	0	5	1	6
Turn-off	12	3	3	2	8
Increase	8	1	0	0	1
Decrease	8	0	3	0	3
Has-value	4	3	1	0	4
lessThan	7	0	0	0	0
greaterThan	8	0	0	0	0
Has-state	1	0	1	0	1
Located-in	6	0	0	0	0
Measured-in	6	0	0	0	0
Detected-in	6	12	0	0	0
Controled-in	6	0	0	0	0

Tableau 5: Résultat de l'extraction des termes dénotant un attribut.

5.7.2 *Protocole d'annotation : Annotation des spécifications utilisateur*

Les spécifications utilisateurs acquises via la plate-forme ont été annotées par quatre annotateurs. Ces derniers possèdent une bonne connaissance des nouvelles technologies sans toutefois être expert dans le domaine des environnements intelligents. L'annotation s'est effectuée à l'aide de l'outil *open source BRAT* (*Brat Rapid Anotation Tool*) [Stenetorp *et al.*, 2012]. *BRAT* offre une interface graphique permettant l'annotation d'entités et de relations entre entités. Le processus d'annotation s'est déroulé en trois phases :

1. annotation d'un sous-ensemble de spécifications qui a permis de préciser les règles d'annotations et de produire une version définitive du guide d'annotation.
2. annotation des spécifications d'exigences d'après les recommandation du guide d'annotation.
3. discussion sur les éventuels désaccords jusqu'au consensus et la production d'une annotation de référence.

Trois types d'annotations ont été définies sous BRAT (cf. figure 25) :

1. les termes référant aux concepts de l'ontologie (en jaune).
2. les termes référant aux propriétés de l'ontologie (en vert).
3. les énoncés contenant une règle de comportement (en violet).

FIGURE 25: Type d'entités définies sous BRAT

La tâche d'annotation a consisté dans un premier temps à annoter les termes référant aux concepts et propriétés de l'ontologie, puis à annoter par des relations les triplets de termes annotés dénotant une instance de propriété. L'annotation a été réalisée en double grâce à des paires d'annotateurs. Les spécifications ont été scindées en deux. Chaque partie a été soumise à deux annotateurs dont un expert [8] et un novice.

La consigne donnée aux annotateurs a été d'annoter les instances de propriétés existantes indépendamment de leur interprétation, c'est-à-dire lorsque la sémantique de celles-ci est modifiée par un modificateur tel que *not* ou *except*. La prise en compte de l'action des modificateurs sur l'interprétation du sens des propriétés est abordée dans nos perspectives.

5.7.3 *Accord inter-annotateurs*

L'annotation humaine étant un processus d'interprétation [Leech, 1997], sa validité ne peut être réellement évaluée. Néanmoins, le calcul des accords inter-annotateurs permet de mesurer la fiabilité et la qualité des annotations produites ainsi que la compréhension par les annotateurs des consignes présentées dans le guide. Elle permet aussi de fixer une borne supérieure aux performances que l'on peut attendre d'un système automatique [Fort et Claveau, 2012]. Calculer l'accord inter-annotateurs (AIA) consiste à déterminer si la corrélation entre les différentes annotations est statistiquement significative. Diverses mesures ont été proposées pour le calcul de cette corrélation. La F-mesure et le *Kappa de Cohen* [Cohen, 1960] sont parmi les plus utilisées. Ces mesures s'appliquent sur les résultats d'accords et de désaccords entre deux annotateurs. Plus exactement, cela consiste à

8. Une personne ayant déjà pratiquée l'annotation

prendre les annotations produites par l'un des annotateurs et d'évaluer les annotations d'un autre annotateur en fonction de celles-ci. Les résultats d'une telle comparaison sont souvent représentées par une *matrice de confusion* (cf. figure 26). Une matrice de confusion décrit les deux cas d'accords possibles (en bleu et en jaune sur la figure) et les deux cas de désaccords possibles (en vert sur la figure) entre les annotations des deux annotateurs. Il existe donc quatre cas de distribution entre les annotations de références et celles de l'hypothèse. Ces derniers sont illustrés en figure 27.

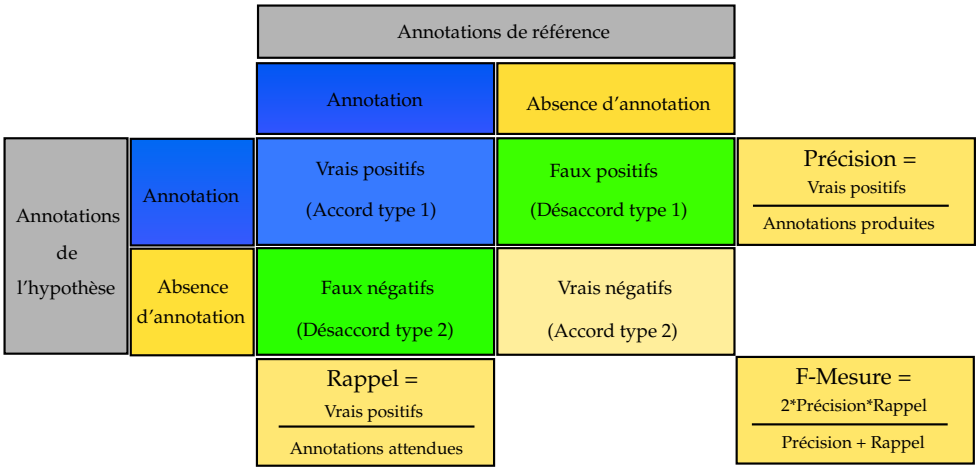


FIGURE 26: Matrice de confusion pour l'annotation d'instances de propriété

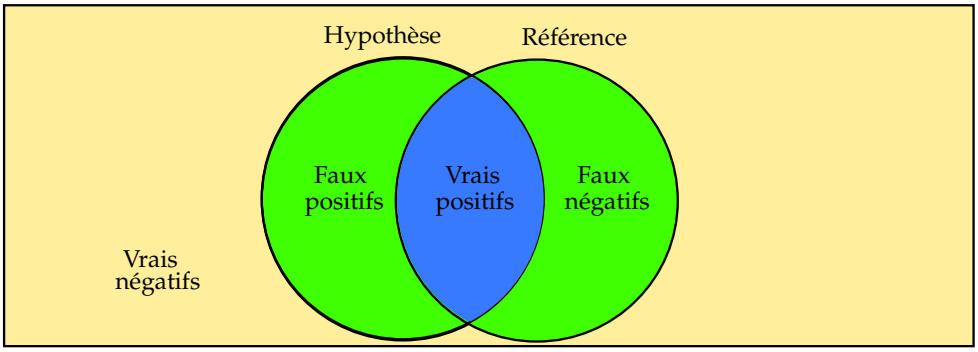


FIGURE 27: Distribution des annotations

Dans le contexte particulier d'annotation de propriétés, seuls trois cas sont déterminés :

1. *vrais positifs* (accord type 1) : nombre des propriétés annotées de la même manière dans la référence et l'évaluation ;
2. *faux positifs* (désaccord type 1) : nombre des propriétés annotées de l'évaluation absentes de la référence.
3. *faux négatifs* (désaccord type 2) : nombre des propriétés annotées dans la référence absentes de l'évaluation.

Le quatrième cas correspond aux *vrais négatifs* (accord de type 2) : les propriétés non annotées dans la référence et l'évaluation. Le nombre de *vrais négatifs* est indéterminé (cf. figure 27).

La difficulté à déterminer ce nombre n'est pas un obstacle pour le calcul de la *F-mesure* qui calcule un ratio entre la précision et le rappel (cf. figure 26 pour les formules). Le *kappa de Cohen* en revanche normalise l'accord observé en fonction d'un accord aléatoire, c'est-à-dire dû au hasard. Pour cela il applique une comparaison des annotations observées avec une *référence aléatoire* portant sur l'ensemble des unités qui auraient pu être annotées, ce qui nécessite d'estimer le nombre de vrais négatifs. La mesure du kappa (k) calcule un rapport entre la probabilité d'accord $Pr(a)$ entre deux annotateurs et la probabilité d'un accord aléatoire $Pr(e)$. Son calcul s'effectue de la manière suivante :

$$\frac{Pr(a)-Pr(e)}{1-Pr(e)}$$

Les valeurs de probabilité $Pr(a)$ et $Pr(e)$ dépendent du nombre de vrais négatifs. L'estimation de ce nombre est une tâche difficile. Une approche pour l'estimer consiste à considérer le nombre d'unités conjointement non annotées comme égal au nombre total d'annotation [Grappy *et al.*, 2010].Une autre approche consiste à considérer l'ensemble des unités (dans notre cas l'ensemble des triplets de termes) du texte comme pouvant être annotées. Cette solution produit une valeur maximale de *kappa* (k) qui comme démontré dans [Grouin *et al.*, 2011] tend à être égale à la F-mesure. La F-Mesure représentant la borne maximale de l'accord inter-annotateur, nous avons choisi de calculer la valeur du *kappa* en estimant le nombre de vrais négatifs à zéro, la valeur de *kappa* représente la borne minimale de l'accord inter-annotateur. La calcul de ces deux mesures permet de borner la valeur de l'accord inter-annotateur.

		A_1	A_1
		Annotée	NA
A_2	Annotée	96	30
A_2	NA	40	-

(A)

		A_3	A_3
		Annotée	NA
A_4	Annotée	174	30
A_4	NA	56	-

(B)

FIGURE 28: Matrice de confusion entre paire d'annotateurs

Les valeurs de Précision (P), de Rappel (R) et de F-Mesure (F-M) ont été calculées à l'aide de l'outil *Brateval*[9], sur la base des informations décrites par les matrices de confusion (A) pour l'accord entre les annotateurs A_1 et A_2 et (B) pour l'accord entre les annotateurs A_3 et

9. https ://bitbucket.org/nicta_biomed/brateval

A_4. De même, les valeurs de probabilité $Pr(a)$ et $Pr(e)$ sont calculées à partir des informations des *matrice de confusion* (A) et (B). Le détail du calcul appliqué est décrit ci-dessous. Les résultats sont présentés dans le tableau 6. La valeur de l'AIA entre A_1 et A_2 appartient à $[0,37, 0,73]$. La valeur de l'AIA entre A_3 et $A4$ appartient à $[0,40, 0,85]$.

AIA entre A_1 et A_2

Probabilité d'accord :
$Pr(a) = 96/166 = 0,5783$

Probabilité d'accord simultané :
$Pr(e) = 0,5783 * 0,5783 = 0,3344$

Calcul du kappa :
$k = \frac{0,5783-0,3344}{1-0,3344} = 0,3664$

AIA entre A_3 et A_4

Probabilité d'accord :
$Pr(a) = 174/260 = 0,6692$

Probabilité d'accord simultané :
$Pr(e) = 0,6692 * 0,6692 = 0,4478$

Calcul du kappa :
$k = \frac{0,6692-0,4478}{1-0,4478} = 0,4009$

	k	P	R	F-M
A_1/A_2	0,37	0,70	0,76	0,73
A_2/A_1	0,37	0,76	0,70	0,73
A_3/A_4	0,40	0,85	0,76	0,85
A_4/A_3	0,40	0,76	0,85	0,85

Tableau 6: Bornes maximale (F-M) et minimale (k) de l'accord inter-annotateur

La valeur de *kappa* telle qu'elle a été calculée représente une valeur plancher pour l'estimation de l'AIA. Néanmoins, l'interprétation de l'accord inter-annotateurs privilégie souvent la *F-Mesure* par rapport au *Kappa de Cohen* dont la valeur fluctue en fonction d'une comparaison avec une référence aléatoire [Alex *et al.*, 2010]. L'interprétation de l'AIA est elle aussi une tâche ardue, différentes propositions ont été faites dans l'état de l'art. L'une des références a pour longtemps été l'échelle proposée par Landis *et al.* [1977], illustrée par la figure 29. Cette échelle considère les valeurs d'AIA au dessus de 0.4 comme acceptables. Cependant, l'état de l'art actuel préconise de suivre une convention plus stricte qui considère les valeurs d'AIA supérieures à 0.8 comme signe d'accord et les valeurs en dessous comme signe de désaccord [Artstein et Poesio, 2008]. En suivant ces préconisations, la valeur de l'accord inter-annotateurs entre A_1 et A_2 reflète un désaccord faible et celui entre A_3 et $A4$ reflète un accord acceptable. L'accord faible entre paire d'annotateurs peut venir du fait d'une diffé-

rence de niveau d'expérience en matière d'annotation entre couple expert-novice ou d'une méconnaissance du domaine des environnements intelligents. Une comparaison des annotations de chaque annotateur avec celles de référence montre un accord plus élevé (cf. tableau 7)

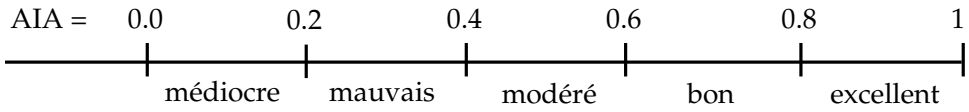


FIGURE 29: Valeur de l'accord inter-annotateurs selon Landis *et al.* [1977]

Le tableau 7 présente le résultat de cette comparaison pour chaque annotateur. Une moyenne des évaluations des quatre annotateurs est donnée en dernière ligne. On y observe une précision et un rappel élevés, particulièrement pour les annotateurs A_2 et A_3, qui contrairement aux annotateurs A_1 et A_4, avaient déjà eu des expériences d'annotation. L'évaluation des annotations humaines permet de fixer une borne supérieure aux performances que l'on peut attendre d'un système automatique.

Annotateur	**Référence**	**Correcte**	**Incorrecte**	**P**	**R**	**F-M**
A_1	130	97	33	0,75	0,75	0,75
A_2	130	117	6	0,93	0,88	0,91
A_3	215	208	8	0,96	0,96	0,96
A_4	215	164	25	0,87	0,76	0,81
Moyenne	**345**	**293**	**36**	**0,89**	**0,85**	**0,86**

Tableau 7: Évaluation des annotations des annotateurs par rapport à celles de référence.

5.7.4 *Évaluation de l'extraction des instances de propriétés*

La configuration automatique du comportement d'un système est une tâche critique. Durant laquelle, l'omission d'une règle de comportement est moins problématique que la création d'une règle incorrecte. Pour un utilisateur, contrôler la justesse de ses propres spécifications d'exigences est moins ardu que contrôler la correction de règles formelles devant les représenter. Aussi, il est important de limiter autant que possible la confrontation de l'utilisateur aux règles de comportement formelles. Limiter cette confrontation repose sur une identification précise des règles spécifiées en langage naturel.

204 règles d'extraction d'instances de propriétés ont été acquises. De la même manière que pour les règles d'extraction de termes (cf. section 5.6.3) ces règles sont issues des 4 chemins de moins de 4 dépendances les plus fréquents et leurs déclinaisons. Ces restrictions ont pour objectif de favoriser la précision du système d'extraction d'instances. Chaque règle est représentée par une fonction Java générée automatiquement à partir de ses caractéristiques (cf. section 3.3.4.5). Ces fonctions font appel à des connaissances ontologiques et terminologiques durant le traitement des spécifications (cf. figure 30). Chacune de ces fonctions représente le contexte syntaxico-sémantique de formulation d'une instance de propriété particulière. Une fonction correspond alors à un ensemble de conditions à vérifier, qui, lorsqu'elles s'appliquent, mènent à la création d'une instance de propriété. L'extraction d'instances de propriétés est effectuée par l'application des fonctions d'extraction d'instances acquises sur l'ensemble des spécifications utilisateur.

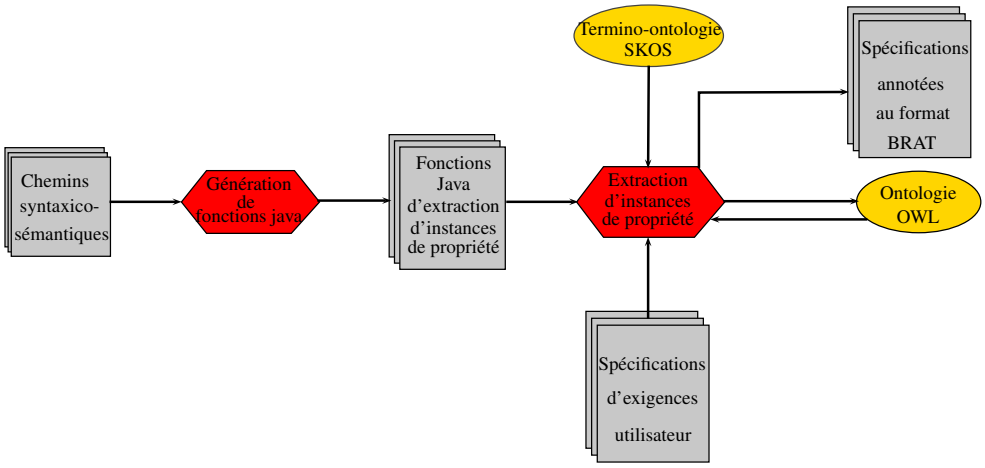


FIGURE 30: Processus d'extraction d'instances de propriétés

Le résultat du processus d'extraction d'instances donne lieu à la création d'un fichier d'annotations au format *BRAT*. L'interprétation de ce fichier via l'interface *BRAT* permet de visualiser les propriétés annotées au sein des spécifications utilisateurs (cf. figure 31). Cela permet une comparaison du résultat de l'annotation automatique à celui de l'annotation de référence, de manière visuelle (via l'interface) et par le calcul des mesures de *précision*, de *rappel* et de *F-Mesure*, effectué de manière automatique par l'outil *Brateval*.

L'évaluation du système d'extraction automatique d'instances de propriété est faite par rapport à une *baseline*. Cette baseline a comme heuristique d'extraire une instance de propriété à chaque fois qu'un triplet de termes la dénote dans une phrase. Les résultats de la baseline sont présentés dans le tableau 8. La ligne *Total* présente le ré-

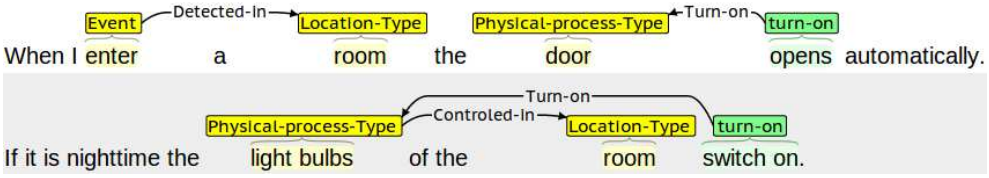


FIGURE 31: Annotations BRAT générées automatiquement

sultat de la *baseline* appliquée sur les termes de la termino-ontologie étendue automatiquement. La ligne *Total + lexique* présente le résultat de la *baseline* appliquée sur la termino-ontologie augmentée par les termes contenus dans les spécifications utilisateurs et non présents dans la termino-ontologie. Dans ces deux lignes, on observe un bon rappel pour une précision très faible. En effet, presque trois quarts des instances de propriété identifiées, le sont incorrectement. L'identification de règles de comportement est fondée sur l'exploitation des instances de propriété extraites. Il est donc nécessaire de privilégier la précision des connaissances extraites sur leur nombre, pour privilégier la correction des règles à leur nombre. La valeur de rappel qui ne dépasse pas les 72% même lorsque l'ensemble de la terminologie est connue, s'explique par l'ambiguïté qui existe entre certaines instances de propriétés et aux erreurs d'identification que cela peut engendrer.

Propriété	**Référence**	**Correcte**	**Incorrecte**	**P**	**R**	**F-M**
Controled-in(phy,loc)	114	86	143	0,37	0,75	0,50
Detected-in(e,loc)	74	54	217	0,20	0,72	0,31
Measured-in(m,loc)	20	10	18	0,36	0,5	0,41
Turn-on(a,phy)	64	39	64	0,38	0,60	0,46
Turn-off(a,phy)	43	32	36	0,47	0,74	0,57
Increase(a,phy)	1	1	1	0,5	1	0,67
Decrease(a,phy)	3	0	0	0	0	0
Located-in(loc,loc)	8	3	131	0,02	0,37	0,42
Has-state(phy,st)	7	1	24	0,04	0,14	0,06
Has-value(m,v)	5	2	7	0,22	0,4	0,29
greaterThan(m,v)	4	1	0	1	0,25	0,4
less-Than(m,v)	2	1	0	1	0,5	0,67
Total	**345**	**230**	**641**	**0,26**	**0,66**	**0,37**
Total + lexique	**345**	**250**	**794**	**0,24**	**0,72**	**0,36**

Tableau 8: Résultats de la baseline pour l'extraction d'instances de propriété.

L'évaluation du système d'extraction d'instances de propriété est présentée en détail dans le tableau 9. On observe une précision globale élevée ($\simeq$ 0.85) proche de celle des annotateurs humains. Cela montre la pertinence des règles acquises. Le rappel obtenu ($\simeq$ 0.41) est relativement bon compte tenu du fait que l'acquisition des règles d'extraction et de la terminologie s'est faite à partir d'un corpus non spécifique au domaine et que les règles acquises ont été appliquées à des spécifications utilisateurs dans un anglais hétérogène. Ces résultats sont en accord avec ceux attendus, i.e. qui privilégient la précision au rappel.

Propriété	**Référence**	**Correcte**	**Incorrecte**	**P**	**R**	**F-M**
Controled-in(phy,loc)	114	23 + (16)	1	0,96	0,34	0,50
Detected-in(e,loc)	74	29	7	0,81	0,39	0,53
Measured-in(m,loc)	20	7	0	1	0,35	0,52
Turn-on(a,phy)	64	35	3	0,92	0,55	0,69
Turn-off(a,phy)	43	26	8	0,76	0,60	0,68
Increase(a,phy)	1	1	0	1	1	1
Decrease(a,phy)	3	2	0	1	0,67	0,80
Located-in(loc,loc)	8	1	3	0,25	0,12	0,17
Has-state(phy,st)	7	1	3	0,25	0,14	0,18
Has-value(m,v)	5	0	0	0	0	0
greaterThan(m,v)	4	0	0	0	0	0
less-Than(m,v)	2	0	0	0	0	0
Total	**345**	**141**	**25**	**0,85**	**0,41**	**0,55**
Total + lexique	**345**	**162**	**30**	**0,84**	**0,47**	**0,60**

Tableau 9: Résultats de l'extraction automatique d'instances de propriété

Afin d'identifier ce qui a fait défaut au niveau du rappel, outre les éventuelles erreurs de l'analyseur syntaxique, la termino-ontologie a été étendue avec les termes présents dans les spécifications utilisateurs et absents de la termino-ontologie. Les règles d'extraction ont alors été appliquées de nouveau sur les spécifications utilisateur. Le résultat de cette application est décrit en ligne *Total + lexique*. On observe un rappel à peine plus élevé ($\simeq$ 0.47). Une nouvelle itération sur l'acquisition de règles d'extraction d'instances de propriété a été lancée en exploitant la termino-ontologie étendue par les termes des spécifications utilisateurs. Suite à cette itération le nombre initial de règles d'extraction n'a pas augmenté. En conclusion, la diversité gram-

maticale du corpus d'entraînement ne semble pas englober en totalité le style grammatical des spécifications utilisateurs. Par exemple, 26 règles ont été acquises pour l'extraction des instances de propriété *Has-value*, *greaterThan* et *less-Than*, néanmoins aucune ne correspondait aux contextes syntaxico-sémantiques des instances de ces propriétés tels qu'ils ont été formulés dans les spécifications utilisateurs. Ce constat motive notre ambition d'exploiter la plate-forme d'acquisition de spécifications utilisateurs pour la constitution d'un corpus de spécifications d'exigences dans le domaine des environnements intelligents. Ce corpus pourra servir à l'amélioration des performances du système d'extraction d'instances de propriété.

Une seconde explication du rappel bas concerne l'identification d'instances de propriétés implicites, notamment pour les instances de propriété *Controled-in*, *Detected-in* et *Measured-in* dont l'écart entre le nombre de référence et celui des instances de propriétés extraites est assez important. En effet, l'annotateur humain a la capacité d'inférer une propriété entre deux entités même lorsque les termes dénotant celles-ci ne sont pas grammaticalement liés.

L'heuristique est alors de lier les entités pouvant être en relation avec une entité de *Location-Type* avec la dernière ou seule occurrence de celle-ci. Dans le souci de privilégier la précision sur le rappel, l'heuristique du système d'extraction est plus restreinte, elle consiste à inférer des instances de propriétés uniquement entre les entités qui participent déjà à une instance de propriété.

C'est ainsi qu'ont été inférées 16 instances de la propriété *Controled-in* supplémentaires aux 23 extraites (cf. ligne 1 tableau 9).

Les annotations incorrectes des sous-propriétés d'*Actuate* (*Turn-on*, *Turn-off*, ...) et celles de la propriété *Has-state* sont corrélées. En effet, même si les deux types de propriétés ne sont pas définies sur les mêmes *domaine* et *image*, les ensembles de triplets de termes qui les dénotent sont très similaires : *Actuate* et *Physical-process* pour *Actuator* [10] et *Physical-process* et *State* pour *Has-state* car les ensembles de termes *Actuate* et *State* partagent plusieurs termes. Lever l'ambiguïté inhérente à ces deux types d'instance de propriété nécessite alors une interprétation dans un contexte plus étendu. Par exemple, la prise en compte de certains termes introducteurs d'une condition peut s'avérer utile. Ces termes peuvent être plus ou moins fréquents. À titre d'exemple les termes « *when* », « *if* » et « *each time* » apparaissent respectivement 42 fois, 26 fois et 2 fois.

Le tableau 10 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*

10. Le triplet correspondant est partiel car la mention aux individus du concept *Actuator* sont le plus souvent implicites

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.

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

Tableau 10: Résultats de la résolution des termes ambigus

5.7.5 *Identification d'instances implicites*

L'originalité de notre 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 11 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. Ce processus d'inférence est effectué lors de l'identification des règles utilisateur décrit dans la section suivante.

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
Zone-of-sensing(s,loc)	28		
Managed-type(a,typ)	28	Actuator	28
Managed-zone(a,loc)	28		

Tableau 11: Instances de propriété et de concept implicites créées au sein de l'ontologie

5.8 IDENTIFICATION ET VÉRIFICATION DES RÈGLES UTILISATEUR

Au sein de l'ontologie, 5 patrons de règles ont été définis. Chacun a pour objectif de guider l'identification d'un type de règles utilisateur en fonction des instances de propriétés extraites des spécifications utilisateur. Nous présentons ci-dessous un exemple de patron de règle de

comportement. Dans cette règle, les éléments en vert correspondent aux instances à identifier à partir de l'analyse des spécifications. Les éléments précédés d'un point d'interrogation correspondent aux variables de la règle. La propriété en rouge est une super-propriété. Elle détermine le type de propriété à identifier à partir de l'analyse des spécifications i.e. ses sous-propriétés.

Detected-in(t,l) Has-type(?ph,t) Perceived-type(?s,t)
Controled-in(p,l) Managed-type(?a,t) Loc-type(?loc,l)
Occurred-in(?ph, ?loc) Managed-zone(?a, ?loc)
Zone-of-sensing(?s, ?loc) ⇒ Actuate(?a,p)

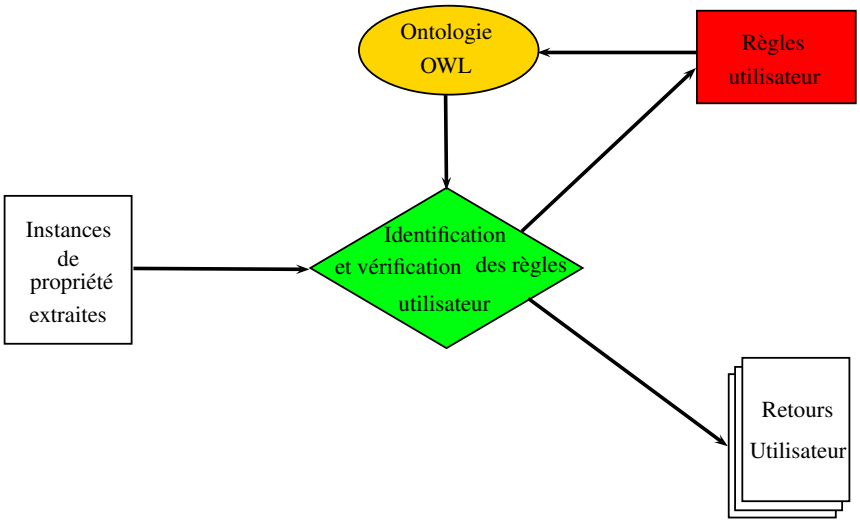


FIGURE 32: Identification des règles utilisateur

La figure 32 illustre le processus d'identification des règles utilisateur à partir des instances de propriétés extraites, guidé par les patrons de règles définis au sein de l'ontologie. Il en résulte un ensemble de règles utilisateur vérifiées créées dans l'ontologie et un fichier contenant les retours utilisateur. Ce fichier contient des informations sur les phrases pouvant contenir des règles en contradiction, incomplètes ou non applicables.

Le tableau 12 détaille les résultats de l'identification de règles utilisateur à partir des instances de propriété extraites. En tout, 28 règles de comportement sur 62 règles annotées ont été identifiées complètement et créées. Parmi elles, 2 règles ont été rejetées, étant en contradiction avec deux règles pré-existantes. 20 règles ont été identifiées comme non applicables car elles possèdent un prédicat dont aucune instance n'est définie au sein de l'ontologie. 10 règles ne sont pas correctement formées : 2 contiennent un prédicat incorrect, 3 un prédicat supplémentaire et 5 un prédicat manquant. L'ensemble des vérifications appliquées aux règles est détaillé ci-après.

Type règle	**Nb**	**Commentaire**
Référence	62	annotées
Identifiée	28	identifiées automatiquement
Incohérente	2	règles incohérentes avec règles existantes
non applicable	20	instances de prédicats non définies dans l'ontologie
Correct	18	tous les prédicats présents et corrects
Incorrect	10	2 prédicats incorrects, 3 prédicats supplémentaires, 5 prédicats manquants

Tableau 12: Résultats de l'identification des règles utilisateur

5.8.1 *Gestion de la cohérence*

Deux règles sont en contradiction lorsqu'elles partagent le même antécédent mais qu'elles concluent sur deux conséquents opposés. Par exemple, la règle *R-84* issue de la phrase numéro 84 [11] a été identifiée comme étant en contradiction avec la règle *R-1* issue de la phrase numéro 1. Ces deux règles ainsi que les phrases qui leurs sont associées, sont alors stockées pour ensuite être soumises à l'utilisateur. La figure 33 illustre les deux règles en contradiction. Les individus en vert correspondent aux connaissances similaires issues des spécifications et les propriétés en rouge 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.

5.8.2 *Gestion des règles incomplètes*

Les règles n'ayant pu être identifiées doivent néanmoins être 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. Elle consiste à renvoyer 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

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

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)
Zone-of-sensing(?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)
Zone-of-sensing(?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 33: Règles utilisateurs extraites et reconnues contradictoires

comme contenant une règle potentiellement incomplète. En tout, 54 phrases ont été stockées comme potentiellement porteuses de règles utilisateur. Elles contiennent en fait toutes les règles utilisateur n'ayant pas été identifiées. 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 (cf. figure 31), 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 système. Par exemple, dans notre modélisation, le terme *nighttime* au même titre que les termes *Sunday* ou *Summer* n'est pas considéré comme un phénomène, caractérisé par un type et une localisation. À notre sens, il dénote un cadre temporel dont la prise en compte est discutée dans nos perspectives.

5.8.3 *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 visible ci-dessous, 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 stockées dans le fichier de *retour utilisateur*. À partir des informations stockées, l'utilisateur peut choisir entre abandonner sa règle ou veiller à ce que la configuration physique de l'environnement intelligent évolue. Les prédicats manquants [12] dans l'ontologie ont été ajoutés afin de permettre l'exécution des règles de comportement sous *Maude*.

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.

5.8.4 *Gestion des règles mal formées*

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

À partir des instances de propriétés extraites [13] des spécifications utilisateurs et des patrons de règles définis, 28 règles utilisateurs ont été identifiées sur un total de 62 règles annotées comme référence, ce qui correspond à un rappel de 0,45. De ces 28 règles, 26 ont été créées

12. Qui décrivent la configuration physique de l'environnement intelligent.
13. Ces instances sont celles extraites à partir de la termino-ontologie étendue.

au sein de l'ontologie car 2 règles ont été identifiées comme étant en contradiction avec deux règles déjà existantes. Les règles mal formées sont au nombre de 10, ce qui correspond à une précision de 0,65. Les règles mal formées ont pour cause l'un des trois cas suivant :

1. 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 5.9) .
2. 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*).
3. une instance de propriété du conséquent non extraite pour 5 règles. Ces 5 règles sont donc incomplètes car une des actions à effectuer n'a pas été identifiée. Cela est dû principalement au faible nombre de chemins qui prennent en charge la conjonction. 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 portant sur « *close the window* » n'a pas été extrait. En effet, lors de l'acquisition des chemins syntaxiques seuls deux occurrences du chemin correspondant au contexte syntaxico-sémantique permettant d'identifier la conjonction en cause ont été trouvés. Compte tenu de sa faible fréquence ce chemin n'a pas été retenu.

La gestion des conjonctions est un point critique dans un processus d'identification de règles de comportement. Afin d'acquérir les chemins portant sur des conjonctions, une itération sur l'acquisition de règles d'extraction a été lancée avec comme paramètre un nombre de dépendances inférieur à 4 pour les chemins syntaxiques et une fréquence d'apparition d'au moins 2 dans le corpus d'apprentissage. Le résultat fut l'acquisition de plus de 4000 fonctions d'extraction d'instances de propriétés. Ce résultat n'est pas raisonnable pour une application devant favoriser la précision.

Une des approches envisagée est de guider l'acquisition de certains types de règles. L'idée est de sélectionner les chemins qui contiennent des conjonctions (ou même des négations) et qui subsument un des chemins les plus fréquents sélectionnés de manière automatique. Cela permettrait de prendre en compte les conjonctions uniquement sur les chemins les plus pertinents de sorte à augmenter le rappel sans perdre en précision. Une seconde approche consisterait à normaliser, ou simplifier les phrases en propageant les conjonctions. Néanmoins, cette seconde approche pourrait introduire une baisse de précision selon la qualité des normalisations effectuées.

5.8.5 *Discussion*

Les résultats présentés dans les tableaux 8 et 9 montrent que le système d'extraction automatique d'instances permet d'obtenir une valeur de rappel (0,47) inférieure à celle de la baseline (0,72) mais qu'il surpasse largement sa précision (0,24) avec une valeur (0,84) qui se rapproche de celle qu'obtient un annotateur humain (cf. tableau 7). Nous avons pu voir qu'en dépit de la précision élevée des instances de propriété extraites, 10 règles sur 28 ont été mal formées. Sur ces 10 règles, 3 règles ont pu être identifiées automatiquement comme erronées. Sept règles ont nécessité un contrôle humain pour être identifiées comme non conformes. Ce résultat montre l'importance d'une extraction de connaissances fiable, à laquelle une approche de type baseline ne convient pas.

Il importe de noter que si le système d'extraction automatique identifie presque la moitié des instances de propriété attendues, l'autre moitié n'est pas oubliée. Comme nous l'avons montré, 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 utilisateurs et du système d'extraction.

5.9 TRANSFORMATION DU MODÈLE ONTOLOGIQUE EN SPÉCIFICATION FORMELLE MAUDE

La transformation du modèle ontologique vers une spécification formelle Maude fait suite à la création du modèle ontologique et à sa vérification. Cette transformation a pour résultat la création des classes, objets, messages et variables Maude nécessaires à la définition des règles de réécriture. Chaque règle de réécriture est créée à partir d'une règle de comportement générique du domaine ou d'une règle utilisateur issue de l'analyse des spécifications. Nous donnons ci-dessous un exemple de règle de réécriture issue d'une règle de comportement générique. Cette règle génère un message *Status-on*(a-11 , p-11) entre un actionneur *a-11* et un processus physique *p-11*. Elle s'applique lorsqu'un actionneur possède comme valeur d'attribut de *Turn-on* un type de processus physique similaire au type (*PP-type*) du processus physique qu'il actionne, i.e. sur lequel il possède une valeur de propriété *Actuate-on*.

Règle de réécriture issue d'une règle générique

```
rl [R-on] :
    < a-11 : Actuator | Turn-on : pt-11, Actuate-on : p-11 >
    < p-11 : Physical-process | PP-type : pt-11 >
    < pt-11 : Physical-process-Type | >
    =>
    < a-11 : Actuator | Turn-on : none, Actuate-on : p-11 >
    < p-11 : Physical-process | PP-type : pt-11 >
    < pt-11 : Physical-process-Type | >
    Status-on(a-11 , p-11) .
```

La première phase de vérification sous Maude correspond à la compilation des spécifications. La compilation permet de valider le typage de la spécification Maude issues du modèle ontologique. Une fois la compilation effectuée, le modèle décrit par les spécifications Maude peut être exploré. La fonction *search* permet cette exploration. Elle permet à partir d'une configuration initiale (cf. section 4.5) d'explorer les différents états du modèle en fonction de l'application des règles de réécriture. Une première étape est donc de constituer une configuration initiale qui représentera l'état du système. 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 du système.

Une vérification importante concerne la cohérence des règles modélisées. Lors du peuplement de l'ontologie, les règles possédant un antécédent similaire pour un conséquent opposé ont été soumises à l'utilisateur. Néanmoins, certaines règles possédant des antécédents différents peuvent mener à des conséquents opposés. L'exploration du modèle permet de vérifier le résultat d'exécution des règles de comportement, c'est-à-dire les états possibles du système. Dans le cas d'un environnement intelligent, on doit par exemple vérifier que deux actions ou messages opposés ne peuvent être envoyés à un même objet de type *Physical-process*.

5.9.1 *Vérification de la cohérence des règles de comportement*

Le temps d'exécution des règles étant potentiellement infini, la vérification de l'état du système est menée par palier. En effet, 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. À 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.

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

La figure 34 présente les deux règles *R-1* et *R-88* dont l'application simultanée peut mener à une incohérence.

5.9.2 *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 au comportement générique attendu, i.e. propre au système. Une vérification importante concerne la complétude du 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 réécritures. En revanche, la requête 4 n'a pas donné de résultat après 5 itérations.

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*, *Zone-of-sensing* : *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*, *Zone-of-sensing* : *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*, *Zone-of-sensing* : *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*, *Zone-of-sensing* : *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 34: Règles de réécriture *R-1* et *R-88*

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 utilisateurs 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 au modèle Maude du système, le système peut, s'il est validé comme cohérent et conforme, être déployé.

5.10 DÉPLOIEMENT DU MODÈLE

La finalité du projet ENVIE VERTE est de permettre le déploiement d'une configuration d'un environnement intelligent utilisant un large éventail de capteurs et d'actionneurs. Pour ce faire, nous avons choisi d'utiliser une machine virtuelle nommée *D-Lite* (Distributed Logic for Internet of Things sErvices) [Cherrier *et al.*, 2011]. D-Lite permet une représentation et un accès universel aux fonctionnalités de divers objets d'un réseau de capteurs (capteurs, actionneurs et processus physiques) indépendamment de leurs caractéristiques techniques. Il standardise la description des fonctionnalités des objets du réseau à l'aide d'automates finis. Ces automates sont complétés par un *alphabet de sortie* qui permet de représenter les communications entre les divers objets. L'automate fini avec sortie constitue alors le code qui lie les fonctionnalités de chaque objet avec chacune de ses utilisations potentielles. D-Lite restreint l'alphabet d'entrée et de sortie des automates aux messages échangés par les objets d'un réseau de capteurs. Les états correspondent aux réactions aux messages reçus. Chaque objet a des entrées et des sorties. La communication entre objets se fait alors par réception et envoi de messages. Lorsqu'un objet reçoit un *message entrant* il doit agir. Agir correspond à émettre un *message sortant*.

D-Lite utilise un langage appelé SALT (Simple Application Logic description using Transducers for Internet of Things) [Cherrier *et al.*,

2013], spécialement développé pour D-Lite. *SALT* permet de décrire des automates finis dans un format XML.

Une fois que les informations spécifiées par l'utilisateur dans sa langue naturelle ont été vérifiées sous OWL et sous Maude, le comportement du système représenté par l'ontologie peut être déployé. Le comportement du système est conditionné par le comportement de ses objets. Le déploiement consiste alors à générer pour chaque objet du réseau un fichier XML. Chaque fichier XML décrit le comportement d'un objet particulier, c'est-à-dire ses messages d'entrée et ses messages de sortie qui viennent en réponse aux premiers. De cette façon, un automate fini avec sorties est défini pour chaque objet.

De façon similaire à la transformation du modèle ontologique vers une spécification formelle Maude, proposée au chapitre précédent (cf. chapitre 4), la transformation proposée fait appel aux fonctions de manipulation de l'ontologie (cf. chapitre 3.1.3). Les connaissances requises correspondent aux individus des concepts *Sensor*, *Actuator* et *Physical-process* ainsi qu'aux relations qu'ils entretiennent.

Deux types de fichiers sont créés. Les fichiers décrivant le comportement des processus physiques de l'environnement intelligent et les fichiers décrivant des objets qui combinent les fonctionnalités (messages) d'un objet de type *Sensor* et d'un objet de type *Actuator*. Pour être combinés, un capteur et un actionneur doivent gérer le même type de phénomène pour une même localisation. Nous présentons ci-dessous un exemple de fichier XML combinant les messages d'entrée d'un capteur et les messages de sortie d'un actionneur. La règle de comportement décrite par ce fichier est délimitée par les balises <*rule*> ... </*rule*>. Elle s'interprète de la manière suivante : un message « *Detect* » entrant (*input*) de type « *movement-in* » déclenche l'envoi d'un message sortant (*output*) « *Status-on* » de type « *light* ».

```
<configuration initial="off">
  <rule>
    <input state="on">
      <message name="Detect" category="movement-in"/>
    </input>
    <output state="off">
      <message name="Status-on" category="light"/>
    </output>
  </rule>
</configuration>
```

Les liens entre les différents objets sont définis dans un second temps, sous une interface graphique. La définition de ces liens doit être établie en fonction de la modélisation du système. Ainsi, en suivant le modèle du comportement d'un environnement intelligent proposé, un objet de type *capteur et actionneur* est lié à l'objet de type

processus physique dont le lien avec l'*actionneur* est défini au sein de l'ontologie par la propriété *Actuate-on*.

5.11 CONCLUSION

Dans ce chapitre, nous avons présenté un cas d'application complet, allant des spécifications d'exigences en langage naturel vers des spécifications formelles (sur différents niveaux de représentation) jusqu'au déploiement du comportement du système vérifié. L'application de notre approche de formalisation de spécification d'exigences en langage naturel au projet ENVIE VERTE a permis de montrer sa faisabilité.

Dans un premier temps, nous avons décrit de quelle manière l'ontologie du comportement d'un environnement intelligent pouvait s'intégrer au modèle générique du comportement du système que nous avons proposé (cf. section 3.1). L'exploitation de l'ontologie que nous avons modélisée et d'une termino-ontologie associée a permis l'acquisition de chemins syntaxiques pertinents entre triplets de termes dénotant une instance de propriété. Les chemins syntaxiques acquis ont servi : 1) à la création de règles d'extraction de termes pour l'extension de la termino-ontologie ; 2) à la création de règles d'extraction d'instances de propriété pour le peuplement de l'ontologie et l'identification de règles utilisateur.
Afin d'adapter les arbres de dépendances générés par l'analyseur syntaxique au traitement des spécifications utilisateur, nous avons décrit un ensemble de prétraitements à effectuer, exploitant les caractéristiques des dépendances ou des chemins syntaxiques qu'elles constituent.

Nous avons évalué notre système par rapport à une approche baseline et aux annotations humaines qui représentent respectivement les bornes inférieure et supérieure des performances attendues. En accord avec nos attentes, la valeur de précision obtenue par notre système a largement surpassé celle de la baseline pour atteindre des performances équivalentes aux performances humaines. Le système étant configuré pour omettre des règles utilisateur plutôt que de prendre le risque de se tromper, la valeur de rappel est inférieure à celle de la baseline. Cette valeur est néanmoins correcte compte tenu du handicap qu'ont constitué : 1) la généralité du corpus d'apprentissage par rapport à la spécificité du domaine ; 2) les différences de styles littéraires entre le corpus d'apprentissage (constitué de e-book en anglais) et les spécifications d'utilisateurs majoritairement francophones. En outre, les informations concernant les instances non extraites ne sont pas oubliées. Elles sont stockées en vue de servir

l'amélioration des spécifications utilisateurs et des performances du système.

Nous avons décrit de quelle manière les patrons de règles représentés au sein de l'ontologie permettent de guider l'identification de règles utilisateur à partir des instances de propriété extraites. L'exemple d'application a pu mettre en lumière les vérifications possibles sur les spécifications ainsi que la pertinence d'utiliser un raisonneur d'OWL et les mécanismes de Maude de façon complémentaire.

L'ensemble des vérifications appliquées tout au long du processus de formalisation des spécifications utilisateur permet de faire remonter différents types d'erreurs. Le maintien du lien entre le texte et les différents niveaux de représentation permet de proposer des retours précis sur les causes de ces erreurs de manière à améliorer les spécifications utilisateur et les performances du processus de formalisation des exigences.

6

CONCLUSION ET PERSPECTIVES

6.1 CONCLUSION

Dans cette étude, nous avons traité la problématique du passage automatique de spécifications en langage naturel vers des spécifications formelles. La difficulté de ce passage tient à l'ampleur du fossé qui sépare ces deux types de spécifications. Les travaux qui se sont intéressés à cette tâche ont démontré la pertinence et la nécessité d'un passage par un modèle intermédiaire, qui aurait pour objet de réduire l'écart entre les deux formalismes. Nous avons aussi opté pour un modèle pivot. Le choix de ce modèle a reposé sur différentes considérations.

Le langage naturel est plus un moyen de communication qu'un moyen de représentation. La compréhension du langage naturel fait appel à des connaissances qui, lorsqu'elles sont communément acceptées, sont tout simplement passées sous silence. Pour rendre compte du sens d'un texte en langage naturel, on ne peut donc se limiter aux connaissances qu'il explicite, mais il faut considérer l'ensemble des connaissances implicites sur lesquelles repose la compréhension du texte. Il s'agit alors de définir ce socle de connaissances, communément acceptées et partagées par la communauté à laquelle s'adresse le texte. C'est l'un des rôles que doit jouer le modèle de représentation intermédiaire.

Pour produire une interprétation sémantique des spécifications d'exigences qui soit non ambiguë, ce modèle doit représenter celles-ci de manière rigoureuse et offrir des mécanismes d'inférence permettant d'établir le lien entre les connaissances du domaine et celles explicitées dans les textes. En outre, ce modèle doit faciliter le passage vers un langage de spécifications formelles.

Notre proposition de modèle pivot s'est donc naturellement orientée vers le formalisme des ontologies dont les rôles sont précisément ceux décrits ci-dessus. Tout au long de nos travaux, nous avons exploré et exploité les possibilités offertes par ce modèle. Cette thèse est le résultat de ces recherches et de nos questionnements, que nous résumons ici.

Notre contribution consiste en trois propositions : a) le développement d'une ontologie générique comme modèle pivot ; b) la spécialisation de l'ontologie à partir de l'analyse des spécifications d'exigences

en langage naturel ; c) la transformation du modèle ontologique obtenu en spécifications formelles Maude.

Nous avons défini un modèle générique pour la représentation du comportement d'un système, de ses composants et de leurs caractéristiques, ainsi que ses règles de comportement génériques. La modélisation proposée permet de guider l'identification de règles de comportement à partir des spécifications d'exigences, en définissant un ensemble de règles de comportement générique du domaine dont les spécialisations correspondent aux règles utilisateurs. La modélisation a été conçue pour aider l'analyse des textes et aussi pour transformer leur interprétation en spécifications formelles quel que soit le domaine modélisé.

De plus, ce modèle fait apparaître la nécessité de distinguer deux types de concepts : individuels et génériques. Cette distinction est le résultat de la coexistence possible de deux types d'individus dans une ontologie : les individus qui représentent des objets du monde et les individus qui encodent des types permettant de caractériser les objets du monde. Dans notre contexte, cela se traduit par une distinction claire entre deux types de concepts : *Composant* et *Type*. Cette distinction permet de rendre compte des différents niveaux de caractérisation des individus au sein du système. Les individus du concept *Composant* et de ses sous-concepts correspondent aux composants du système, alors que les individus du concept *Type* et de ses sous-concepts correspondent à des types permettant de caractériser les composants du système. La prise en compte de ces deux sortes de concepts possède plusieurs avantage. Tout d'abord, lors du développement de l'ontologie, elle peut permettre de guider les choix entre concept et individu. Elle permet aussi de soulever les différences entre les modes de reconnaissance des individus de chaque type de concepts à partir de l'analyse des textes et de leur identification au sein de l'ontologie. En particulier, l'identification des individus que nous nommons *référents individuels* ne peut s'effectuer directement par leurs mentions dans les textes. Ces derniers ne peuvent être identifiés qu'à partir de leurs propriétés définissantes. Enfin, cette distinction permet une traduction directe vers un langage de spécifications formelles.

Notre approche pour le peuplement d'ontologie est centrée sur l'identification de mentions d'instances de propriété dans les textes, ce qui permet, grâce à la définition des propriétés définissantes de chaque concept individuel, d'identifier chaque individu issu de l'analyse des spécifications à l'aide d'un raisonnement sur l'ontologie. Les mentions d'instances de propriétés sont reconnues comme des triplets de termes dans les textes. Ces triplets sont identifiés par des règles d'extraction acquises automatiquement et fondées sur les chemins syntaxiques récurrents reliant les termes qui désignent les instances

de concept et de propriété. Ces règles exploitent des connaissances lexicales, syntaxiques et sémantiques. Elles identifient les instances de propriété, même si la mention de l'instance de leur domaine ou image est implicite. La définition précise et restreinte des domaine et image de chaque propriété permet de diminuer, voire de lever l'ambiguïté des termes extraits, en capturant le contexte sémantique dans lequel ils apparaissent. Guider l'identification des individus par l'identification d'instances de propriété permet alors de les reconnaître non pas seulement à partir de leur contexte linguistique mais aussi et surtout dans leur contexte sémantique. Ainsi, leur identification ne s'appuie pas uniquement sur une reconnaissance lexicale mais prend aussi en compte les rôles sémantiques des individus.

Nous avons décrit l'étendue du pouvoir d'inférence et de vérification d'OWL lorsque l'investissement sémantique est substantiel. OWL permet alors de classer les individus et de les identifier de manière unique lors du processus de peuplement automatique et de valider la cohérence du résultat du peuplement avant la transformation du modèle ontologique en spécifications formelles.

Notre approche de transformation de spécifications OWL en spécifications Maude est fondée sur la transformation entre modèles. Elle repose sur l'exploitation de fonctions de manipulation d'éléments ontologiques. Ces fonctions permettent d'accéder directement aux connaissances nécessaires à la création des spécifications Maude. Le modèle ontologique proposé offre une correspondance directe entre les éléments de l'ontologie et ceux d'un module orienté objet Maude. En outre, le formalisme logique du modèle résultant du peuplement de l'ontologie autorise une traduction automatique vers le langage de spécifications Maude. Nous avons montré que cette traduction permet effectivement de compléter les vérifications permises par OWL.

L'approche de formalisation mise en place a été conçue de manière indépendante du domaine d'application et de sorte à être adaptable à d'autres langues. Les adaptations à réaliser pour une application à un autre domaine que celui décrit au chapitre 5 ou à une autre langue que l'anglais sont décrites dans nos perspectives.

6.2 PERSPECTIVES

Dans cette section, nous décrivons nos perspectives à court terme, puis à plus long terme. Nos perspectives à court terme auront pour objet de répondre à chacune des questions suivantes : a) comment prouver de manière formelle que ce que l'on vérifie sous Maude est bien ce qui est spécifié sous OWL, ou en d'autres termes que la transformation du modèle ontologique en une spécification formelle Maude préserve la sémantique initiale ? b) comment appliquer notre approche

de modélisation à un autre domaine et l'adapter à une autre langue ? c) comment 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.

Nos perspectives à plus long terme portent sur l'interprétation de textes plus complexes et l'application de notre approche à des cas réels de configuration d'un environnement intelligent allant jusqu'à son déploiement.

6.2.1 *Perspectives à court terme*

Préservation de la sémantique entre OWL et Maude

Le passage des spécifications OWL aux spécifications Maude a été établi de manière empirique. Afin d'avoir une plus grande assurance de la correction de cette transformation, c'est à dire que ce que l'on a vérifié est bien ce que l'on a spécifié, notre prochain objectif est de vérifier celle-ci de manière formelle. Une telle vérification repose sur la définition de sémantiques formelles décrivant le langage source OWL et le langage cible Maude. Il s'agit alors de vérifier que certaines propriétés sont préservées d'un modèle à un autre. Décrire la sémantique formelle d'un langage consiste à définir un modèle mathématique, du sens de toute spécification écrite dans ce langage. Cela nécessite l'usage d'outils permettant de tester et d'exécuter les sémantiques formalisées [Tollitte *et al.*, 2012] et de vérifier formellement des propriétés de préservation sémantique [Blazy, 2008].

Application à un autre domaine

Le premier pan de notre contribution consiste en une méthode de modélisation des connaissances conçue pour offrir un cadre de représentation générique du comportement d'un système décrit à base de règles éventuellement concurrentes. Cette méthode est pensée pour pouvoir s'appliquer à différents domaines si tant est que l'on fournisse une ontologie formelle et une termino-ontologie associée. L'ontologie fournie doit pouvoir s'imbriquer comme sous-partie de l'ontologie générique. Cela repose sur une spécialisation de l'ontologie qui consiste à : 1) distinguer les concepts sous-concepts de *Composant* et les concepts sous-concepts de *Type* ; 2) distinguer les propriétés de type relation de celles de type attribut ; 3) distinguer les attributs dynamiques des attributs statiques 4) définir pour chaque concept, ses propriétés définissantes ; 5) définir les règles d'inférence SWRL permettant d'inférer les individus identiques.

Adaptation à d'autres langues.

Dans notre approche, la partie linguistique est liée essentiellement à trois ressources : un analyseur syntaxique, une ressource termino-ontologique et un corpus de textes en langage naturel. L'adaptation à une autre langue dépend donc de la disponibilité de ces trois ressources. Plusieurs langues disposent d'un analyseur syntaxique permettant de générer des arbres de dépendances syntaxiques qui constituent l'entrée de notre processus d'acquisition de règles d'extraction de termes et d'instances de propriétés. Une termino-ontologie fait le lien entre les classes sémantiques d'une ontologie et leur formulation dans la langue associée. En SKOS, il est possible d'indiquer pour chaque terme défini, la langue à laquelle celui-ci appartient. Cela permet de définir pour une même classe sémantique les déclinaisons dans différentes langues de son terme préféré et de ses termes alternatifs. Le passage à une autre langue ne nécessite donc pas la création d'une nouvelle terminologie mais uniquement la traduction des termes déjà existants. En outre, les corpus utilisés ne nécessitent aucune pré-annotation.

Évolution de la plate-forme d'acquisition de spécifications utilisateur

La plate-forme actuelle permet a un utilisateur de spécifier ses besoins en langage naturel. L'évolution de cette plate-forme à pour objectif d'aider de manière interactive un utilisateur à produire des spécifications d'exigences complètes et cohérentes. Pour cela, nous projetons d'incorporer à l'interface web existante les fonctionnalités de l'outil de formalisation et de vérifications des exigences. L'interface pourra alors, après vérification des exigences fournir à l'utilisateur des directives visant à préciser ou à corriger ses exigences. L'interaction avec l'utilisateur permettra dans le même temps d'améliorer les performances de l'outil de formalisation et de vérification des exigences.

6.2.2 *Perspectives à plus long terme*

Aller plus loin dans l'interprétation des textes

L'un des premiers aspects à prendre en compte est sans nul doute la négation. L'interprétation de la négation est essentiellement un problème linguistique. En effet, ses modes de formulation dépendent de la langue utilisée et peuvent selon la langue, apparaître sous des formes très variées telles que préposition, adverbe, adjectif, pronom, conjonction ou encore en préfixe ou en suffixe des mots. Gérer ce phénomène demande de faire appel à des ressources linguistiques autres

que la termino-ontologie de sorte à représenter une plus large quantité d'informations spécifiques à la langue à traiter.

Un deuxième aspect non moins important consisterait à étendre l'identification des règles de comportement au delà de la phrase. En effet, l'identification des règles de comportement proposée ne considère que celles formulées au sein d'une même phrase. Cette restriction a deux raisons. Premièrement, ce cas est le plus fréquent et deuxièmement, identifier une règle de comportement formulée sur plusieurs phrases demande d'étendre l'analyse des textes au niveau discursif. Tenir compte de la structure discursive des spécifications d'exigences, nécessite de pouvoir gérer des phénomènes linguistiques aussi complexes que les cadres de discours [Charolles, 1997] ou la co-référence. Un cadre de discours est une unité textuelle qui s'étend au delà de la phrase et dont les éléments entretiennent un même rapport avec un critère sémantique d'interprétation spécifié par une expression introductrice de cadre. Il s'agit alors d'être en mesure d'interpréter une unité textuelle en fonction du cadre dans lequel elle est énoncée. Par exemple dans un cadre *temporel* tel que « *En hiver, ...* » ou dans un cadre *spatial* tel que « *Dans les chambres : ...* ». La co-référence est le phénomène qui consiste à désigner le même objet par plusieurs expressions différentes du texte.

Peupler l'ontologie à partir des spécifications permet de construire une représentation sémantique globale du texte et non des phrases isolées. Une piste réside alors dans l'exploitation de l'ensemble des connaissances de l'ontologie pour étendre les inférences faites au niveau de la phrase, au niveau du texte tout entier.

Déploiement d'une configuration d'un environnement intelligent

Nous avons vu en section 5 qu'il est possible à partir des connaissances spécifiées au sein de l'ontologie de générer des fichiers XML qui décrivent des automates à états finis représentant le comportement de chaque composant d'un environnement intelligent. Notre ambition est de pouvoir construire un modèle complet allant d'une spécification de comportement d'un cas d'utilisation réel d'un environnement intelligent jusqu'à son déploiement.

BIBLIOGRAPHIE

Message sequence chart (msc), itu-ts, geneva, 1996. http://www.itu.int/ITU-T/2001-2004/com17/languages/Z.120AnnB-0498.pdf. accédé : Décembre 2013. (Cited on page 19.)

Object constraint language (ocl), object management group (omg). http://www.omg.org/spec/OCL. accédé : Décembre 2013. (Cited on page 20.)

Unified modeling language (uml), object management group (omg). http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF. accédé : Décembre 2013. (Cited on page 19.)

Russell J. Abbott : Program design by informal english descriptions. *Commun. ACM*, 26(11):882–894, 1983. (Cited on pages 13 et 68.)

Taiseera Hazeem Al Balushi, Pedro R. Falcone Sampaio et Pericles Loucopoulos : Eliciting and prioritizing quality requirements supported by ontologies : a case study using the elicito framework and tool. *Expert Systems*, 30(2):129–151, 2013. (Cited on page 21.)

Harith Alani, Sanghee Kim, David E. Millard, Mark J. Weal, Wendy Hall, Paul H. Lewis et Nigel Shadbolt : Web based knowledge extraction and consolidation for automatic ontology instantiation. *In Knowledge Capture (K-Cap'03), Workshop on Knowledge Markup and Semantic Annotation*, 2003. (Cited on page 73.)

Abdaladhem Albreshne, Ayoub Ait Lahcen et Jacques Pasquier : A framework and its associated process-oriented domain specific language for managing smart residential environments. *International Journal of Smart Home.*, 7(6):377–392, 2013. (Cited on page 155.)

Bea Alex, Claire Grover, Rongzhou Shen et Mijail Kabadjov : Agile corpus annotation in practice : An overview of manual and automatic annotation of cvs. *In Proceedings of the Fourth Linguistic Annotation Workshop*, pages 29–37, 2010. (Cited on page 171.)

Florence Amardeilh : Ontopop or how to annotate documents and populate ontologies from texts. *In ESWC 2006 Workshop on Mastering the Gap : From Information Extraction to Semantic Representation*, volume 187, 2006. (Cited on page 68.)

Florence AMARDEILH : *Web Sémantique et Informatique Linguistique : propositions méthodologiques et réalisation d'une plateforme logicielle*. Thèse de doctorat, Université de Nanterre-Paris X, 2007. (Cited on pages 73 et 75.)

Vincenzo AMBRIOLA et Vincenzo GERVASI : An environment for cooperative construction of natural-language requirement bases. *In Proceedings of the 8th International Conference on Software Engineering Environments (SEE '97)*, pages 124–130, 1997a. (Cited on pages 12 et 13.)

Vincenzo AMBRIOLA et Vincenzo GERVASI : Processing natural language requirements. *In In Proceedings of ASE 1997*, pages 36–45. IEEE Press, 1997b. (Cited on page 12.)

H. H. AMMAR, W. ABDELMOEZ et M. S. HAMDI : Software engineering using artificial intelligence techniques : Current state and open problems. *In ICCIT*, page 24–29, 2012. (Cited on page 14.)

Raghu ANANTHARANGACHAR, Srinivasan RAMANI et S RAJAGOPALAN : Ontology guided information extraction from unstructured text. *International Journal of Web & Semantic Technology (IJWesT)*, 4(1), 2013. (Cited on pages 60 et 72.)

Kyriakos ANASTASAKIS, Behzad BORDBAR, Geri GEORG et Indrakshi RAY : Uml2alloy : A challenging model transformation. *In 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS*, pages 436–450, 2007. (Cited on pages 18, 20, 126 et 129.)

Derek ANDREWS et P. GIBBINS : *An Introduction to Formal Methods of Software Development*. An Introduction to Formal Methods of Software Development. McGraw-Hill Education, 1988. (Cited on page 12.)

Grigoris ANTONIOU, , Grigoris ANTONIOU, Grigoris ANTONIOU, Frank Van HARMELEN et Frank Van HARMELEN : Web ontology language : Owl. *In Handbook on Ontologies in Information Systems*, pages 67–92, 2003. (Cited on page 38.)

Ron ARTSTEIN et Massimo POESIO : Inter-coder agreement for computational linguistics. *Comput. Linguist.*, 34(4):555–596, 2008. (Cited on page 171.)

Sasikanth AVANCHA, Chintan PATEL et Anupam JOSHI : Ontology-driven adaptive sensor networks. *In Proceedings of the First Annual*

International Conference on Mobile and Ubiquitous Systems : Networking and Services (MobiQuitous'04), pages 194–202, 2004. (Cited on page 155.)

Fadi Badra, Sylvie Despres et Rim Djedidi : Ontology and lexicon : The missing link. *In Workshop Proceedings of the 9th International Conference on Terminology and Artificial Intelligence*, pages 16–18, 2011. (Cited on page 59.)

I. Bajwa, Lee M. B. et Behzad : Sbvr business rules generation from natural language specification. *In AAAI 2011 Spring Symposium – AI for Business Agility*, pages 2–8, 2011. (Cited on pages 18 et 22.)

Imran Sarwar Bajwa, Behzad Bordbar, Mark Lee et Kyriakos Anastasakis : Nl2alloy : A tool to generate alloy from nl constraints. *Journal of Digital Information Management*, 10(6), 2012. (Cited on pages 17 et 21.)

Imran Sarwar Bajwa, Behzad Bordbar et Mark G. Lee : Ocl constraints generation from natural language specification. *In Proceedings of the 2010 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC '10)*, 2010. (Cited on pages 17 et 19.)

ImranS. Bajwa et MarkG. Lee : Transformation rules for translating business rules to ocl constraints. *In Modelling Foundations and Applications*, volume 6698, pages 132–143. 2011. (Cited on page 18.)

Amaia Bernaras, Iñaki Laresgoiti et Jose Manuel Corera : Building and reusing ontologies for electrical network applications. *In ECAI96*, pages 298–302, 1996. (Cited on page 34.)

Sandrine Blazy : *Sémantiques formelles.* Habilitation à diriger des recherches., Université d'Evry-Val d'Essonne, 2008. (Cited on page 194.)

Viktor Boer, Maarten Someren et BobJ. Wielinga : Relation instantiation for ontology population using the web. *In KI 2006 : Advances in Artificial Intelligence*, volume 4314, pages 202–213. 2007. (Cited on pages 58, 63 et 65.)

Grady Booch : *Object-Oriented Analysis and Design with Applications (3rd Edition)*. 2004. (Cited on page 68.)

Artur Boronat et José Meseguer : An algebraic semantics for mof. *Formal Aspects of Computing*, 22(3-4):269–296, 2010. (Cited on pages 125 et 127.)

J. Bouaud, B. Bachimont, J. Charlet et P. Zweigenbaum : Acquisition and structuring of an ontology within conceptual graphs. *In ICCS*, volume 94, pages 1–25, 1994. (Cited on pages 28 et 52.)

Ilyès Boukhari, Ladjel Bellatreche et Stéphane Jean : An ontological pivot model to interoperate heterogeneous user requirements. *In Proceedings of the 5th international conference on Leveraging Applications of Formal Methods, Verification and Validation : applications and case studies - Volume Part II*, ISoLA'12, pages 344–358, 2012. (Cited on page 22.)

Paolo Bouquet, Fausto Giunchiglia, Frank Van Harmelen, Luciano Serafini et Heiner Stuckenschmidt : C-owl : Contextualizing ontologies. *In Journal Of Web Semantics*, pages 164–179, 2003. (Cited on page 34.)

Ronald J. Brachman et Hector J. Levesque : The tractability of subsumption in frame-based description languages. *In 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pages 34–37, 1984. (Cited on page 38.)

M. Brill, W. Damm, J. Klose, B. Westphal et H. Wittke : Live sequence charts : An introduction to lines, arrows, and strange boxes in the context of formal verification. *SoftSpez Final Report*, pages 374–399, 2004. (Cited on page 19.)

B. Bryant et B.-S. Lee : Two-level grammar as an object-oriented requirements specification language. *In Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS '02)*, volume 9, page 280, 2002. (Cited on pages 67 et 68.)

Paul Buitelaar et Bernardo Magnini : Ontology learning from text : An overview. *In Ontology Learning from Text : Methods, Applications and Evaluation*, pages 3–12, 2005. (Cited on page 46.)

Paul Buitelaar, Daniel Olejnik et Michael Sintek : A protégé plug-in for ontology extraction from text based on linguistic analysis. *In The Semantic Web : Research and Applications*, volume 3053, pages 31–44. 2004. (Cited on page 48.)

Paul Buitelaar et Melanie Siegel : Ontology-based information extraction with soba. *In Proc. of the International Conference on Language Resources and Evaluation (LREC '06)*, pages 2321–2324, 2006. (Cited on page 73.)

Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka, Jr. et Tom M. Mitchell : Coupled semi-supervised

learning for information extraction. *In Proceedings of the Third ACM International Conference on Web Search and Data Mining*, pages 101–110, 2010. (Cited on pages 62 et 114.)

Verónica Castañeda, Luciana C Ballejos et Maria Laura Caliusco : Improving the quality of software requirements specifications with semantic web technologies. *In WER*, 2012. (Cited on page 22.)

Silvana Castano, Alfio Ferrara, Stefano Montanelli et Davide Lorusso : Instance matching for ontology population. *In SEBD*, pages 121–132, 2008. (Cited on pages 61, 72 et 73.)

David Celjuska et Dr. Maria Vargas-vera : Ontosophie : A semi-automatic system for ontology population from text. *In International Conference on Natural Language Processing (ICON)*, 2004. (Cited on pages 60, 65 et 74.)

David Celjuska, Maria Vargas-Vera, Dávid Čeljuska, Dr. Maria Vargas-vera et Dr. Maria Vargas-vera : Semi-automatic population of ontologies from text. *In Paralic J., Rauber A. (eds.) Workshop on Data Analysis WDA-2004*, 2004. (Cited on page 74.)

P. Ceravolo, C. Fugazza et M. Leida : Modeling semantics of business rules. *In EcoSystems, Digital Technologies Conference. DEST 07*, pages 171–176, 2007. (Cited on page 18.)

Wafa Chama, Raida Elmansouri et Allaoua Chaoui : A modeling and verification approach based on graph transformation. *In Lecture Notes on Software Engineering vol. 1, no. 1*, volume 1, pages 39–43, 2013. (Cited on pages 20, 126, 127 et 129.)

Olivier Chapelle, Bernhard Schlkopf et Alexander Zien : *Semi-Supervised Learning*. 1st édition, 2010. (Cited on page 64.)

Jean Charlet, B. Bachimont et R Troncy : Ontologies pour le web sémantique. *Information - Interaction - Intelligence*, 2004. (Cited on page 34.)

Thierry Charnois : *Détection d'anomalies sémantiques dans les textes de spécifications - application au problème des interactions de service de télécommunication*. Thèse de doctorat, Université de Paris 13, 1999. (Cited on page 18.)

Michel Charolles : L'encadrement du discours univers, champs, domaines et espaces. *Cahier de Recherche Linguistique*, (2):1–73, 1997. (Cited on page 196.)

S. Cherrier, Y.M. Ghamri-Doudane, S. Lohier et G. Roussel : D-lite : Distributed logic for internet of things services. *In Internet of Things (iThings/CPSCom), 4th International Conference on Cyber, Physical and Social Computing*, pages 16–24, 2011. (Cited on page 187.)

Sylvain Cherrier, Yacine Ghamri-Doudane, Stephane Lohier et Gilles Roussel : Salt : a simple application logic description using transducers for internet of things. *In IEEE International Conference on Communications ICC*, page 8, 2013. (Cited on page 187.)

Amina Chniti, Patrick Albert et Jean Charlet : Gestion de la cohérence des règles métier éditées à partir d'ontologies owl. *In Actes de IC2011*, pages 589–606, 2012. (Cited on page 22.)

Noam Chomsky : A review of b. f. skinner's verbal behavior. *Language*, 35(1):26–58, 1959. (Cited on page 54.)

Noam Chomsky et Françoise Dubois-Charlier : Un compte rendu du « comportement verbal » de bf skinner. *Langages*, (16):16–49, 1969. (Cited on pages 54 et 158.)

A. Cimatti, E. Clarke, F. Giunchiglia et M. Roveri : Nusmv : a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2:2000, 2000. (Cited on page 129.)

Philip Cimiano et Johanna Völker : Text2onto - a framework for ontology learning and data-driven change discovery. *In Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3513, pages 227–238, 2005. (Cited on pages 48 et 65.)

Philipp Cimiano, Paul Buitelaar, John McCrae et Michael Sintek : Lexinfo : A declarative model for the lexicon-ontology interface. *Web Semantics : Science, Services and Agents on the World Wide Web*, 9(1):29–51, 2011. (Cited on page 59.)

Philipp Cimiano, Alexander Mädche, Steffen Staab et Johanna Völker : Ontology learning. *In Handbook on Ontologies*, pages 245–267. 2009. (Cited on page 48.)

Philipp Cimiano et Johanna Völker : Towards large-scale, open-domain and ontology-based named entity classification. Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP), pages 166–172, 2005. (Cited on pages 60, 62, 65 et 68.)

Fabio Ciravegna : (lp)2, an adaptive algorithm for information extraction from web-related texts. *In In Proceedings of the IJCAI-2001*

Workshop on Adaptive Text Extraction and Mining, 2001. (Cited on page 69.)

M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer et J. F. Quesada : Maude : specification and programming in rewriting logic. *Theor. Comput. Sci.*, 285(2):187–243, 2002. (Cited on pages 97 et 124.)

Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Marti-Oliet, José Meseguer et Carolyn Talcott : Maude manual (version 2.6). http ://maude.cs.uiuc.edu/maude2-manual/maude-manual.pdf, 2011. accédé : Décembre 2013. (Cited on pages 124 et 131.)

Manuel Clavel, Francisco Durán, Joe Hendrix, Salvador Lucas, José Meseguer et Peter Ölveczky : The maude formal tool environment. *In* Till Mossakowski, Ugo Montanari et Magne Haveraaen, éditeurs : *Algebra and Coalgebra in Computer Science*, volume 4624, pages 173–178. 2007. (Cited on page 125.)

Manuel Clavel et Marina Egea : Equational specification of uml+ocl static class diagrams. http://maude.sip.ucm.es/~clavel/pubs/clavel-egea06a.pdf, 2006a. accédé : Janvier 2013. (Cited on page 127.)

Manuel Clavel et Marina Egea : Itp/ocl : A rewriting-based validation tool for uml+ocl static class diagrams. *In 11th International Conference on Algebraic Methodology and Software Technology AMAST'06*, 2006b. (Cited on pages 20, 125, 126 et 127.)

Jacob Cohen : A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960. (Cited on page 168.)

Hamish Cunningham, Hamish Cunningham, Diana Maynard, Diana Maynard, Valentin Tablan et Valentin Tablan : Jape : a java annotation patterns engine. 2nd edition. Rapport technique, Department of Computer Science, University of Sheffield, Sheffield,UK, 2000. (Cited on page 67.)

Werner Damm et David Harel : Lscs : Breathing life into message sequence charts. *Form. Methods Syst. Des.*, 19(1):45–80, juillet 2001. (Cited on page 19.)

Sourish Dasgupta, Ankur Padia, Kushal Shah, Rupali KaPatel et Prasenjit Majumder : Dlolis-a : Description logic based text ontology learning. *CoRR*, abs/1303.5929, 2013. (Cited on page 47.)

Marie-Catherine De Marneffe et Christopher D Manning : Stanford typed dependencies manual. http://nlp.stanford.edu/downloads/dependencies_manual.pdf, 2008. accédé : Novembre 2013. (Cited on page 161.)

Gerard de Melo : Not quite the same : Identity constraints for the web of linked data. *In Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013. (Cited on page 74.)

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer et Richard Harshman : Indexing by latent semantic analysis. *Journal of the american society for information science*, 41(6): 391–407, 1990. (Cited on page 60.)

Kathrin Dentler, Ronald Cornet, Annette ten Teije et Nicolette de Keizer : Comparison of reasoners for large ontologies in the owl 2 el profile. *Semantic Web*, 2(2):71–87, 2011. (Cited on page 92.)

Rolf Drechsler, Mathias Soeken et Robert Wille : Formal specification level : Towards verification-driven design based on natural language processing. *In Specification and Design Languages (FDL)*, pages 53–58, 2012. (Cited on page 19.)

Francisco Durán, Martin Gogolla et Manuel Roldán : Tracing properties of uml and ocl models with maude. *Proceedings of International Workshop on Algebraic Methods in Model-Based Software Engineering (AMMSE 2011), Electronic Proceedings in Theoretical Computer Science*, 2011. (Cited on page 128.)

Francisco Duràn et Manuel Roldàn : Validating ocl constraints on maude prototypes of uml models. http ://maude.lcc.uma.es/mOdCL/docs/TR/DynamicValidation/TR-Prototyping.pdf, 2012. accédé : Décembre 2013. (Cited on pages 20, 126, 127 et 129.)

Steven Eker, José Meseguer et Ambarish Sridharanarayanan : The maude {LTL} model checker. *Electronic Notes in Theoretical Computer Science*, 71:162 – 187, 2004. (Cited on page 125.)

Daniel Elenius, Grit Denker et Mark-Oliver Stehr : A semantic web reasoner for rules, equations and constraints. *In* Diego Calvanese et Georg Lausen, éditeurs : *Web Reasoning and Rule Systems*, volume 5341, pages 135–149. 2008. (Cited on pages 125 et 128.)

David W. Embley : Toward semantic understanding : An approach based on information extraction ontologies. *In Proceedings of the 15th Australasian Database Conference - Volume 27*, ADC '04, pages 3–12, 2004. (Cited on page 63.)

A. FANTECHI, S. GNESI, G. RISTORI, M. CARENINI, M. VANOCCHI et P. MORESCHINI : Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, 4 (3):243–263, 1994. (Cited on page 66.)

S. FARFELEDER, T. MOSER, A. KRALL, T. STÅLHANE, H. ZOJER et C. PANIS : Dodt : Increasing requirements formalism using domain ontologies for improved embedded systems development. *In Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 271–274, 2011. (Cited on pages 18 et 22.)

Carla FARIA, Ivo SERRA et Rosario GIRARDI : A domain-independent process for automatic ontology population from text. *Science of Computer Programming*, (0), 2013. (Cited on pages 62 et 68.)

John Rupert FIRTH : A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis (special volume of the Philological Society)*, 1952-59:1–32, 1957. (Cited on page 104.)

Karën FORT et Vincent CLAVEAU : Annotation manuelle de matchs de foot : Oh la la la ! l'accord inter-annotateurs ! et c'est le but ! *In Actes de la conférence conjointe JEP-TALN-RECITAL 2012, volume 2 : TALN*, pages 383–390, 2012. (Cited on page 168.)

Blaz FORTUNA, Marko GROBELNIK et Dunja MLADENIC : Ontogen : semi-automatic ontology editor. *In Proceedings of the 2007 conference on Human interface : Part II*, pages 309–318, 2007. (Cited on page 49.)

Alain-Jérôme FOUGÈRES et Philippe TRIGANO : Rédaction de spécifications formelles : Élaboration à partir des spécifications écrites en langage naturel. *Cognito - Cahiers Romans de Sciences Cognitives*, 1 (8):29–36, 1997. (Cited on pages 13 et 21.)

M. D. FRASER, K. KUMAR et V. K. VAISHNAVI : Informal and formal requirements specification languages : Bridging the gap. *IEEE Trans. Softw. Eng.*, 17(5):454–466, 1991. (Cited on pages 9, 13 et 16.)

Norbert E. FUCHS, Kaarel KALJURAND et Tobias KUHN : Reasoning web. chapitre Attempto Controlled English for Knowledge Representation, pages 104–124. 2008. (Cited on page 16.)

Norbert E. FUCHS et Rolf SCHWITTER : Attempto controlled natural language for requirements specifications. *In Proc. Seventh Intl. Logic Programming Symp. Workshop Logic Programming Environments*, pages 25–32, 1995. (Cited on page 17.)

Katrin Fundel, Robert Küffner et Ralf Zimmer : Relex–relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2007. (Cited on page 69.)

Frédéric Fürst et Francky Trichet : Heavyweight ontology engineering. *In* Robert Meersman, Zahir Tari et Pilar Herrero, éditeurs : *On the Move to Meaningful Internet Systems 2006 : OTM 2006 Workshops*, volume 4277, pages 38–39. 2006. (Cited on page 47.)

Patrice Gagnon, Farid Mokhati et Mourad Badri : Applying model checking to concurrent uml models. *Journal of Object Technology*, 7 (1):59–84, 2008. (Cited on page 20.)

Françoise Gayral, Daniel Kayser et François Lévy : Logique et sémantique du langage naturel : modèles et interprétation. *intellectica*, 2(23):303–325, 1996. (Cited on pages 15 et 73.)

Vincenzo Gervasi et Didar Zowghi : Reasoning about inconsistencies in natural language requirements. *ACM Trans. Softw. Eng. Methodol.*, 14(3):277–330, 2005. (Cited on page 17.)

Akila Ghersedine, Patrice Buche, Juliette Dibie-Barthélemy, Nathalie Hermandez et Mouna Kamel : Extraction de relations n-aires interphrastiques guidée par une rto. *In CORIA*, pages 179–190, 2012. (Cited on page 59.)

Martin Giese et Rogardt Heldal : From informal to formal specifications in uml. *In Proc. of UML2004, Lisbon, volume 3273 of LNCS*, pages 197–211, 2004. (Cited on page 20.)

Claudio Giuliano et Alfio Gliozzo : Instance based lexical entailment for ontology population. *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 248–256, 2007. (Cited on page 65.)

Claudio Giuliano et Alfio Gliozzo : Instance-based ontology population exploiting named-entity substitution. *In Proceedings of the 22Nd International Conference on Computational Linguistics (COLING '08)*, volume 1, pages 265–272, 2008. (Cited on pages 58, 60, 66 et 72.)

Michal Gordon et David Harel : Generating executable scenarios from natural language. *In Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing '09)*, pages 456–467, 2009. (Cited on pages 8, 17 et 63.)

Daniel Gouadec : *Terminologie : constitution des données*. AFNOR : Association française de normalisation, 1990. (Cited on page 54.)

Arnaud GRAPPY, Brigitte GRAU, Olivier FERRET, Cyril GROUIN, Véronique MORICEAU, Isabelle ROBBA, Xavier TANNIER, Anne VILNAT et Vincent BARBIER : A corpus for studying full answer justification. *In Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, 2010. (Cited on page 170.)

Cyril GROUIN, Sophie ROSSET, Pierre ZWEIGENBAUM, Karën FORT, Olivier GALIBERT et Ludovic QUINTARD : Proposal for an Extension of Traditional Named Entitites : from Guidelines to Evaluation, an Overview. *In Proceedings of the Fifth ACL Linguistic Annotation Workshop*, pages 92–100, 2011. (Cited on page 170.)

Thomas R. GRUBER : A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199 – 220, 1993. (Cited on page 79.)

Thomas R. GRUBER : Toward principles for the design of ontologies used for knowledge sharing ? *International Journal of Human-Computer Studies*, 43(5–6):907 – 928, 1995. (Cited on pages 34 et 46.)

Nicola GUARINO : Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, 46(2-3):293–310, 1997. (Cited on pages 35 et 52.)

Nicola GUARINO : Some ontological principles for designing upper level lexical resources. *In Proceedings of the First International Conference on Language Resources and Evaluation*, 1998. (Cited on page 46.)

Nicola GUARINO, Daniel OBERLE et Steffen STAAB : What is an ontology ? *In Handbook on Ontologies*. 2009. (Cited on page 23.)

Mahdi GUEFFAZ : *ScaleSem : Model Checking et Web Sémantique*. These, Université de Bourgogne, 2012. (Cited on pages 125 et 129.)

Mahdi GUEFFAZ, Sylvain RAMPACEK et Christophe NICOLLE : Rdf2nµsmv : mapping semantic graphs to nµsmv model checker. *In AFIN 2011, The Third International Conference on Advances in Future Internet*, pages 49–53, 2011a. (Cited on page 129.)

Mahdi GUEFFAZ, Sylvain RAMPACEK et Christophe NICOLLE : Rdf2spin : Mapping semantic graphs to spin model checker. *In Digital Information and Communication Technology and Its Applications*, volume 167, pages 591–598. 2011b. (Cited on page 129.)

Abdoulaye GUISSÉ : *Une plateforme d'aide à l'acquisition et à la maintenance des règles métier à partir de textes règlementaires*. These, UNIVERSITÉ PARIS 13 - SORBONNE PARIS CITÉ, 2013. (Cited on pages 17 et 63.)

Abdoulaye Guissé, François Lévy et Adeline Nazarenko : 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)*, pages 77–91, 2012. (Cited on pages 13 et 17.)

S. Gulan, S. Johr, R. Kretschmer, S. Rieger et M. Ditze : Graphical modelling meets formal methods. *In Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, pages 716–721, 2013. (Cited on pages 20 et 126.)

Volker Haarslev et Ralf Möller : Racer system description. *In Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 701–706, 2001. (Cited on page 41.)

Zellig Harris : *The Form of Information in Science : Analysis of an Immunology Sublanguage*. Boston Studies in the Philosophy of Science. 1989. (Cited on page 69.)

Takaaki Hasegawa, Satoshi Sekine et Ralph Grishman : Discovering relations among named entities from large corpora. *In Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, 2004. (Cited on pages 68 et 69.)

Nathalie Hernandez : *Ontologies de domaine pour la modélisation du contexte en Recherche d'Information*. Thèse de doctorat, Université Paul Sabatier, 2005. (Cited on pages 47 et 57.)

S Hoefler : The syntax of attempto controlled english : An abstract grammar for ace 4.0. Rapport technique, 2004. (Cited on page 17.)

Gerard Holzmann : *Spin Model Checker, the : Primer and Reference Manual*. Addison-Wesley Professional, first édition, 2003. (Cited on page 129.)

Matthew Horridge et Sean Bechhofer : The owl api : A java api for working with owl 2 ontologies. *In OWLED*, pages 11–21, 2009. (Cited on page 88.)

Ian Horrocks : The fact system. *In Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312. 1998. (Cited on page 41.)

Ian Horrocks : Owl : A description logic based ontology language. *In Principles and Practice of Constraint Programming - CP 2005*, volume 3709, pages 5–8. Springer Berlin Heidelberg, 2005. (Cited on pages 38 et 41.)

Ian Horrocks, Peter F. Patel-Schneider et Frank van Harmelen : From {SHIQ} and {RDF} to owl : the making of a web ontology language. *Web Semantics : Science, Services and Agents on the World Wide Web*, 1(1):7–26, 2003. (Cited on page 39.)

Ning Huang, Xiaojuan Wang et Camilo Rocha : Formal semantics of owl-s with rewrite logic. *JSEA*, 2(1):25–33, 2009. (Cited on pages 125, 128 et 129.)

Duvravka Ilic : Deriving formal specifications from informal requirements. *In Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC '07)*, volume 1, pages 145–152, 2007. (Cited on page 8.)

Magda Ilieva et Harold Boley : Representing textual requirements as graphical natural language for uml diagram generation (seke'08). pages 478–483, 2008. (Cited on page 19.)

H. Kaiya et M. Saeki : Using domain ontology as domain knowledge for requirements elicitation. *In Requirements Engineering, 14th IEEE International Conference*, pages 189–198, 2006. (Cited on pages 18 et 21.)

J. Karpovic et L. Nemuraite : Transforming sbvr business semantics into web ontology language owl2 :main concepts. *In Proc. 17th International Conference on Information and Software Technologies*, 2011. (Cited on page 18.)

Ruth M. Kempson : Semantics, pragmatics, and natural ?language interpretation. *In The Handbook of Contemporary Semantic Theory*, pages 561–598. 1996. (Cited on page 15.)

O. Keszocze, M. Soeken, E. Kuksa et R. Drechsler : Lips : An ide for model driven engineering based on natural language processing. *In Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 31–38, 2013. (Cited on page 20.)

Asad Masood Khattak, Phan Tran Ho Truc, Le Xuan Hung, La The Vinh, Viet-Hung Dang, Donghai Guan, Zeeshan Pervez, Manhyung Han, Sungyoung Lee et Young-Koo Lee : Towards smart homes using low level sensory data. *Sensors*, 11(12):11581–11604, 2011. (Cited on page 155.)

Khaled Khelif, Rose Dieng-Kuntz et Pascal Barbry : An ontology-based approach to support text mining and information retrieval in the biological domain. *JUCS*, 13(12):1881–1907, 2007. (Cited on page 67.)

Sanghee Kim, Harith Alani, Wendy Hall, Paul H. Lewis, David E. Millard, Nigel R. Shadbolt et Mark J. Weal : Artequakt : Generating tailored biographies with automatically annotated fragments from the web. *In Semantic Authoring, Annotation and Knowledge Markup (SAAKM) Workshop at the 15th European Conference on Artificial Intelligence (ECAI'02)*, pages 1–6, 2002. (Cited on page 73.)

Hasan Kitapci et Barry W. Boehm : Using a hybrid method for formalizing informal stakeholder requirements inputs. *In Proceedings of the Fourth Internationa Workshop on Comparative Evaluation in Requirements Engineering (CERE '06)*, pages 48–59, 2006. (Cited on page 13.)

Michael Klein, Andreas Schmidt et Rolf Lauer : 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*, 2007. (Cited on page 155.)

Leonid Kof : 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*, NLDB'09, pages 79–90, 2009. (Cited on pages 19, 20 et 64.)

S Koide : *Theory and Implementation of Object Oriented Semantic Web Language*. Thèse de doctorat, Dept. Informatics School of Multidisciplinary Sciences, The Graduate Univ. for Ad- vanced Studies (SOKENDAI), 2010. (Cited on page 124.)

Seiji Koide, Jans Aasman et Steve Haflich : Owl vs. object orientated programming. *In International Semantic Web Conference, Workshop1 : Semantic Web Enabled Software Engineering*, 2005. (Cited on page 124.)

Sven J. Körner et Torben Brumm : Resi - a natural language specification improver. *IEEE Sixth International Conference on Semantic Computing*, 0:1–8, 2009. (Cited on pages 20 et 22.)

Sven J. Körner et Torben Brumm : Natural language specification improvement with ontologies. *International Journal of Semantic Computing (IJSC)*, 3(4):445–470, 2010. (Cited on pages 18 et 21.)

Zornitsa Kozareva, Ellen Riloff et Eduard H. Hovy : Semantic class learning from the web with hyponym pattern linkage graphs. *In ACL*, pages 1048–1056, 2008. (Cited on page 74.)

Seif Eddine Kramdi, Ollivier Haemmerlé et Nathalie Hernandez : Approche générique pour l'extraction de relations à partir de textes.

In Actes des 2oes Journées Francophones d'Ingénierie des Connaissances, pages 97–108, 2009. (Cited on page 69.)

Tobias Kuhn : A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, 2014. (Cited on page 11.)

R. Laleau et A. Mammar : An automatic generation of b specifications from well-defined uml notations for database applications. *In Int. Symp. on Programming Systems, Alger, Algérie*, 2001. (Cited on pages 20 et 126.)

Mathias Landhäusser, SvenJ. Körner et WalterF. Tichy : From requirements to uml models and back : how automatic processing of text can support requirements engineering. *Software Quality Journal*, pages 1–29, 2013. (Cited on pages 18 et 19.)

J Richard Landis, Gary G Koch *et al.* : The measurement of observer agreement for categorical data. *biometrics*, 33(1):159–174, 1977. (Cited on pages xi, 171 et 172.)

Hung Ledang et Jeanine Souquières : Formalizing uml behavioral diagrams with b. *In Tenth OOPSLA Workshop on Behavioral Semantics : Back to Basics*, 2001. (Cited on pages 20 et 126.)

Beum-Seuk Lee et Barrett R. Bryant : Contextual knowledge representation for requirements documents in natural language. *In Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, pages 370–374, 2002. (Cited on pages 15 et 69.)

Yuqin Lee et Wenyun Zhao : Domain requirements elicitation and analysis - an ontology-based approach. *In Proceedings of the 6th International Conference on Computational Science - Volume Part IV*, pages 805–813, 2006. (Cited on page 21.)

Geoffrey Leech : Introducing corpus annotation. *In Corpus annotation : Linguistic information from computer text corpora*, pages 1–18, 1997. (Cited on page 168.)

Daniel Leitao, Dante Torres et Flávia de Almeida Barros : Nlforspec : Translating natural language descriptions into formal test case specifications. *In SEKE*, pages 129–134, 2007. (Cited on page 66.)

Dekang Lin et Patrick Pantel : Discovery of inference rules for question answering. *Natural Language Engineering*, 2001. (Cited on pages 68 et 69.)

Jinxin Lin, Mark S. Fox et Taner Bilgic : A requirement ontology for engineering design. *Concurrent Engineering : Research and Applications*, 4:279–291, 1996. (Cited on pages 18 et 21.)

Alessio Lomuscio et Franco Raimondi : Mcmas : A model checker for multi-agent systems. *In Proceedings of TACAS 2006*, pages 450–454, 2006. (Cited on page 128.)

Alessio Lomuscio et Monika Solanki : Towards an agent based approach for verification of owl-s process models. *In The Semantic Web : Research and Applications*, volume 5554, pages 578–592. 2009. (Cited on page 128.)

Alexander Maedche, Er Maedche et Raphael Volz : The ontology extraction maintenance framework text-to-onto. *In Proceedings of the ICDM'01 Workshop on Integrating Data Mining and Knowledge Management*, 2001. (Cited on page 48.)

Bernardo Magnini, Emanuele Pianta, Octavian Popescu et Manuela Speranza : Ontology population from textual mentions : Task definition and benchmark, 2006. (Cited on pages 60, 61 et 68.)

Jawad Makki, Anne-Marie Alquier et Violaine Prince : Ontology Population via NLP Techniques in Risk Management. *International Journal of Humanities and Social Sciences*, 3(3):212–217, 2009. (Cited on pages 54, 68, 69, 72 et 99.)

Frank Manola et Eric Miller : Rdf primer, 2004. (Cited on page 38.)

Narciso Martí-Oliet et José Meseguer : Rewriting logic as a logical and semantic framework. *In* J. Meseguer, éditeur : *Handbook of Philosophical Logic*, volume 9 de *Handbook of Philosophical Logic*, pages 1–87. Elsevier Science Publishers, 2002. (Cited on page 125.)

Diana Maynard, Adam Funk et Wim Peters : Sprat : a tool for automatic semantic pattern-based ontology population. *In International Conference for Digital Libraries and the Semantic Web*, 2009. (Cited on page 60.)

Diana Maynard, Yaoyong Li et Wim Peters : Nlp techniques for term extraction and ontology population. *In Proceedings of the 2008 conference on Ontology Learning and Population : Bridging the Gap between Text and Knowledge*, pages 107–127, 2008. (Cited on pages 62 et 68.)

Diana Maynard, Horacio Saggion, Milena Yankova, Kalina Bontcheva et Wim Peters : Natural language technology for information integration in business intelligence. *In Business Information Systems*, volume 4439, pages 366–380. 2007. (Cited on page 73.)

John McCrae, Dennis Spohr et Philipp Cimiano : Linking lexical resources and ontologies on the semantic web with llemon. *In Proceedings of the 8th extended semantic web conference on The semantic web (ESWC'11) : research and applications - Volume Part I*, pages 245–259, 2011. (Cited on page 59.)

Luke K. McDowell et Michael Cafarella : Ontology-driven, unsupervised instance population. *Web Semant.*, 6(3):218–236, 2008. (Cited on page 64.)

José Meseguer : A logical theory of concurrent objects. *OOPSLA/ECOOP '90 Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, 25(10):101–115, 1990. (Cited on page 124.)

José Meseguer : Twenty years of rewriting logic. *In Rewriting Logic and Its Applications*, volume 6381, pages 15–17. Springer Berlin Heidelberg, 2010. (Cited on page 125.)

F. Meziane et S. Vadera : Artificial intelligence in software engineering current developments and future prospects. *In Artificial Intelligence Applications for Improved Software Engineering Development : New Prospects*, page 24–29, 2010. (Cited on page 14.)

Luisa Mich, Mariangela Franch et Pierluigi Inverardi : Market research for requirements analysis using linguistic tools. *Requirement Engineering*, 9(1):40–56, 2004. (Cited on pages 12 et 16.)

Marvin Minsky : A framework for representing knowledge. Rapport technique, 1974. (Cited on page 38.)

Farid Mokhati et Mourad Badri : Generating maude specifications from uml use case diagrams. *Journal of Object Technology*, 8(2):119–136, 2009. (Cited on pages 20, 126 et 127.)

Noureddine Mokhtari : *Extraction et exploitation d'annotations sémantiques contextuelles à partir de texte*. Thèse de doctorat, Université de NICE-SOPHIA ANTIPOLIS École doctorale STIC Sciences et Technologie de l'Information et de la Communication, 2010. (Cited on page 62.)

Thibault Mondary, Sylvie Després, Adeline Nazarenko et Sylvie Szulman : Construction d'ontologies à partir de textes : la phase de conceptualisation. *In Actes des 19èmes Journées Francophones d'Ingénierie des Connaissances (IC'08)*, pages 87–98, 2008. (Cited on pages 48, 61 et 68.)

Yayoi NAKAMURA-DELLOYE : Named entity extraction for ontology enrichment. *In IPSJ Special Interest Group - Information Fundamentals and Access Technologies (IFAT)*, 2011. (Cited on pages 68 et 69.)

Yayoi NAKAMURA-DELLOYE et Rosa STERN : Extraction de relations et de patrons de relations entre entités nommées en vue de l'enrichissement d'une ontologie. *In TOTh 2011 : Terminologie & Ontologie : Théories et Applications*, 2011. (Cited on pages 68, 70 et 71.)

Yayoi NAKAMURA-DELLOYE et Éric VILLEMONTE DE LA CLERGERIE : Exploitation de résultats d'analyse syntaxique pour extraction semi-supervisée des chemins de relations. *In TALN 2010*, 2010. (Cited on page 70.)

Robert NECHES, Richard FIKES, Tim FININ, Tom GRUBER, Ramesh PATIL, Ted SENATOR et William R. SWARTOUT : Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991. (Cited on page 34.)

Hong Phuong NGUYEN : *DÉRIVATION DES PÉCIFICATIONS FORMELLES B À PARTIR DE SPÉCIFICATIONS SEMI-FORMELLES*. Thèse de doctorat, Centre d'Études et De Recherche en Informatique du CNAM (CEDRIC), 1998. (Cited on pages 20 et 126.)

P.B.F. NJONKO et W. EL ABED : From natural language business requirements to executable models via sbvr. *In Systems and Informatics (ICSAI), 2012 International Conference on*, 2012. (Cited on page 17.)

OMG : Semantics of business vocabulary and business rules(sbvr), v1.0, 2008. (Cited on page 17.)

Inah OMORONYIA, Guttorm SINDRE, Tor STÅLHANE, Stefan BIFFL, Thomas MOSER et Wikan SUNINDYO : A domain ontology building process for guiding requirements elicitation. *In Requirements Engineering : Foundation for Software Quality*, volume 6182, pages 188–202. 2010. (Cited on page 21.)

Nouha OMRANE, Adeline NAZARENKO, Sylvie SZULMAN *et al.* : Comment guider le travail de normalisation terminologique ? *23es Journées francophones d'Ingénierie des Connaissances*, 2012. (Cited on page 60.)

Wenbo PANG, Xiaozhong FAN, Jiangde YU et Yuxiang JIA : Chinese semantic class learning from web based on concept-level characteristics. *In PACLIC*, pages 415–424, 2009. (Cited on page 74.)

Florent PERES, Jing YANG et Mohamed GHAZEL : A formal framework for the formalization of informal requirements. *International Journal of Soft Computing and Software Engineering [JSCSE]*, pages 14–27, 2012. (Cited on pages 8, 12 et 13.)

Georgios PETASIS, Vangelis KARKALETSIS, Georgios PALIOURAS, Anastasia KRITHARA et Elias ZAVITSANOS : Ontology population and enrichment : State of the art. *In Knowledge-Driven Multimedia Information Extraction and Ontology Evolution'11*, 2011. (Cited on pages xi, 54, 55, 56 et 61.)

Georgios PETASIS, Ralf MÖLLER et Vangelis KARKALETSIS : Boemie : Reasoning-based information extraction. *In NLPAR@LPNMR*, pages 60–75, 2013. (Cited on pages 50, 51, 57, 72 et 73.)

Roberto POLI : Ontology for knowledge organization. *Advances in Knowledge Organization*, 5:313–319, 1996. (Cited on page 52.)

José E RIVERA, Francisco DURÁN et Antonio VALLECILLO : Formal specification and analysis of domain specific models using maude. *Simulation*, 85(11-12):778–792, 2009. (Cited on pages 125 et 127.)

Christophe ROCHE : Le terme et le concept : fondements d'une ontoterminologie. *Terminologie & Ontologie : Theories et applications (TOTh)*, 2008. (Cited on page 59.)

Christophe ROCHE, Marie CALBERG-CHALLOT, Luc DAMAS et Philippe ROUARD : Ontoterminology : A new paradigm for terminology. *In International Conference on Knowledge Engineering and Ontology Development*, pages 321–326, 2009. (Cited on page 59.)

Manuel ROLDAN et Francisco DURAN : Dynamic validation of ocl constraints with modcl. *In Workshop on OCL and Textual Modelling*, volume 44, 2010. (Cited on pages 20, 126 et 127.)

José Raúl ROMERO, José Eduardo RIVERA, Francisco DURÁN et Antonio VALLECILLO : Formal and tool support for model driven engineering with maude. *Journal of Object Technology*, 6(9):187–207, 2007. (Cited on pages 125 et 127.)

J. M. RUIZ-MARTÌNEZ, J. A. MIÑARRO-GIMÉNEZ, D. CASTELLANOS-NIEVES, F. GARCÌA-SÁNCHEZ et R. VALENCIA-GARCÌA : Ontology population : an application for the e-tourism domain. *International Journal of Innovative Computing ; Information and Control (IJICIC)*, pages 6115–6134, 2011. (Cited on pages 54, 57, 60, 62, 67, 68, 72 et 74.)

Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik et Douglas D. Edwards : *Artificial intelligence : a modern approach*. Prentice-Hall, Inc., 1996. (Cited on pages 14 et 48.)

D.J Russomanno, C Kothari et O Thomas : Sensor ontologies : from shallow to deep models. *In System Theory, SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium*, pages 107–112, 2005. (Cited on page 155.)

Vlad Rusu : Embedding domain-specific modeling languages into maude specifications. *Software and Systems Modeling*, 12(4):847–869, 2013. (Cited on page 125.)

Vlad Rusu et Manuel Clavel : Vérification d'invariants pour des systèmes spécifiés en logique de réécriture. *Studia Informatica Universalis*, 7(2), 2009. (Cited on page 136.)

Driss Sadoun : Peuplement d'une ontologie modélisant le comportement d'un environnement intelligent guidé par l'extraction d'instances de relations. *In Actes de la 14e Rencontres des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RECITAL'2012)*, pages 281–294, 2012. (Cited on page 29.)

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane et Brigitte Grau : An ontology for the conceptualization of an intelligent environment and its operation. *In 10th Mexican International Conference on Artificial Intelligence (MICAI)*, pages 16–22, 2011. (Cited on page 23.)

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane et Brigitte Grau : Formalisation en owl pour vérifier les spécifications d'un environnement intelligent. *In Actes de la conférence RFIA*, 2012. (Cited on page 27.)

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane et Brigitte Grau : From natural language requirements to formal specification using an ontology. *In 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2013a. (Cited on page 28.)

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane et Brigitte Grau : Peuplement d'une ontologie guidé par l'identification d'instances de propriété. *In Actes de la 10e conférence TIA*, 2013b. (Cited on pages 28 et 29.)

M. Saeki, H. Horai et H. Enomoto : Software development process from natural language specification. *In Software Engineering, 1989. 11th International Conference on*, pages 64–73, 1989. (Cited on page 13.)

Rolf SCHWITTER : Controlled natural languages for knowledge representation. *In Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10) : Posters*, pages 1113–1121, 2010. (Cited on pages 11 et 16.)

Rukman SENANAYAKE, Grit DENKER et Jon PEARCE : Towards integrated specification and analysis of machine-readable policies using maude 1. *In Workshop on Semantic Web and Policies at International Semantic Web Conference*, 2005. (Cited on pages 125 et 128.)

Masayuki SHIBAOKA, Haruhiko KAIYA et Motoshi SAEKI : Goore : Goal-oriented and ontology driven requirements elicitation method. *In Advances in Conceptual Modeling – Foundations and Applications*, volume 4802, pages 225–234. 2007. (Cited on page 21.)

E SIRIN, B PARSIA, B GRAU, A KALYANPUR et Y KATZ : Pellet : A practical owl-dl reasoner. *Web Semantics Science Services and Agents on the World Wide Web*, 5(2):51–53, 2007. (Cited on page 41.)

R.L. SMITH, G.S. AVRUNIN, L.A. CLARKE et L.J. OSTERWEIL : Propel : an approach supporting property elucidation. *In Proceedings of the 24rd International Conference on Software Engineering (ICSE'02)*, pages 11–21, 2002. (Cited on page 17.)

Stephen SODERLAND, David FISHER, Jonathan ASELTINE et Wendy LEHNERT : Crystal inducing a conceptual dictionary. *In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995. (Cited on page 65.)

Ian SOMMERVILLE et Pete SAWYER : *Requirements Engineering : A Good Practice Guide*. John Wiley & Sons, Inc., 1st édition, 1997. (Cited on page 8.)

Yingjie SONG, Rong CHEN et Yaqing LIU : A non-standard approach for the owl ontologies checking and reasoning. *Journal of Computers (JCP)*, 7(10):2454–2461, 2012. (Cited on pages 125, 129 et 140.)

Amina SOUAG. : Vers une nouvelle génération de définition des exigences de sécurité fondée sur l'utilisation des ontologies. *In INFORSID*, pages 583–590, 2012. (Cited on page 22.)

J. F. SOWA : *Conceptual Structures : Information Processing in Mind and Machine*. Addison-Wesley Longman Publishing Co., Inc., 1984. (Cited on page 51.)

Pontus STENETORP, Sampo PYYSALO, Goran TOPIĆ, Tomoko OHTA, Sophia ANANIADOU et Jun'ichi TSUJII : Brat : a web-based tool for nlp-assisted text annotation. *In Proceedings of the Demonstrations at*

the 13th Conference of the European Chapter of the Association for Computational Linguistics, pages 102–107, 2012. (Cited on page 167.)

Kiyoshi Sudo, Satoshi Sekine et Ralph Grishman : Automatic pattern acquisition for japanese information extraction. *In Proceedings of the first international conference on Human language technology research*, 2001. (Cited on pages 68, 69 et 70.)

Algirdas Sukys, , Lina Nemuraite, , Bronius Paradauskas, et Edvinas Sinkevicius : Sbvr based representation of sparql queries and swrl rules for analyzing semantic relations. *In Proceedings of the First International Conference on Business Intelligence and Technology (BUSTECH'11)*, pages 1–6, 2011. (Cited on page 18.)

Algirdas Sukys, Lina Nemuraite, Bronius Paradauskas et Edvinas Sinkevicius : Transformation framework for sbvr based semantic queries in business information systems. *In The Second International Conference on Business Intelligence and Technology*, pages 19–24, 2012. (Cited on page 18.)

Xinruo Sun, Haofen Wang et Yong Yu : Sapop : Semiautomatic framework for practical ontology population from structured knowledge bases. *In Semantic Web and Web Science*, pages 181–186. 2013. (Cited on pages 65 et 72.)

B. Swartout, P. Ramesh, K. Knight et T. Russ : Toward distributed use of large-scale ontologies. *AAAI Symposium on Ontological Engineering*, 1997. (Cited on page 34.)

Sylvie Szulman : Une nouvelle version de l'outil terminae de construction de ressources termino-ontologiques. *In 22èmes journées francophones d'Ingénierie des Connaissances (poster)*, page 3 pages, 2011. (Cited on pages 48, 49 et 90.)

Sylvie Szulman, Nathalie Aussenac-Gilles, Jean Charlet, Adeline Nazarenko, Eric Sardet et H.V. Teguiak : Dafoe : A platform for building ontologies from texts. *In Proceedings of the EKAW2010 Poster and Demo Track*, 2010. (Cited on page 59.)

Naimdjon Takhirov, Fabien Duchateau et Trond Aalberg : An evidence-based verification approach to extract entities and relations for knowledge base population. *In International Semantic Web Conference*, volume 7649, pages 575–590, 2012. (Cited on pages 72, 74, 99 et 108.)

Hristo Tanev et Bernardo Magnini : Weakly supervised approaches for ontology population. *In Proceedings of the 2008 conference on Onto-*

logy Learning and Population : Bridging the Gap between Text and Knowledge, pages 129–143, 2008. (Cited on pages 61, 62, 65 et 68.)

Theerayut THONGKRAU et Pattarachai LALITROJWONG : Ontopop : An ontology population system for the semantic web. *IEICE Transactions*, 95(4):921–931, 2012. (Cited on pages 58, 60, 61, 64 et 72.)

Walter F. TICHY et Sven J. KOERNER : Text to software : developing tools to close the gaps in software engineering. *In Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER'10)*, pages 379–384, 2010. (Cited on pages 19 et 20.)

Pierre-Nicolas TOLLITTE, David DELAHAYE et Catherine DUBOIS : Producing certified functional code from inductive specifications. *In Certified Programs and Proofs*, volume 7679, pages 76–91. 2012. (Cited on page 194.)

Ninh Thuan TRUONG : *Utilisation de B pour la vérification de spécifications UML et le développement formel orienté objet*. Thèse de doctorat, Université Nancy II, 2006. (Cited on page 126.)

M. USCHOLD et M. GRUNINGER : Ontologies : Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996. (Cited on page 21.)

Michael USCHOLD et Michael GRUNINGER : Ontologies and semantics for seamless connectivity. *ACM SIGMOD Record*, 33(4):58–64, 2004. (Cited on page 22.)

Mike USCHOLD et Martin KING : Towards a methodology for building ontologies. *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995. (Cited on page 46.)

S. VADERA et F. MEZIANE : From english to formal specifications. *The Computer Journal*, 37(9):753–763, 1994. (Cited on page 16.)

AlexandrosG. VALARAKOS, Georgios PALIOURAS, Vangelis KARKALETSIS et George VOUROS : Enhancing ontological knowledge through ontology population and enrichment. *In Engineering Knowledge in the Age of the Semantic Web*, volume 3257, pages 144–156. Springer Berlin Heidelberg, 2004. (Cited on page 65.)

G. VAN HEIJST, A. Th. SCHREIBER et B. J. WIELINGA : Using explicit ontologies in kbs development. *International Journal of Human-Computer Studies*, 46(2-3):183–292, 1997. (Cited on page 52.)

A. van WIJNGAARDEN : *Orthogonal Design and Description of a Formal Language*. Stichting Mathematisch Centrum, 1965. (Cited on page 67.)

Alberto Verdejo, Narciso Martí-Oliet, Tomás Robles, Joaquín Salvachúa, Luis Llana et Margarita Bradley : Transforming information in rdf to rewriting logic. *In Formal Methods for Open Object-Based Distributed Systems*, volume 3535, pages 227–242. 2005. (Cited on pages 125 et 128.)

Feng Wang et Kenneth J. Turner : 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*, ICOST '09, pages 66–73, 2009. (Cited on page 155.)

Richard C. Wang et William W. Cohen : Automatic set instance extraction using the web. *In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, volume 1, pages 441–449, 2009. (Cited on page 74.)

K. E. Wiegers : When telepathy won't do : Requirements engineering key practices. *Cutter IT Journal*, 2000. (Cited on page 9.)

Daya C. Wimalasuriya et Dejing Dou : Ontology-based information extraction : An introduction and a survey of current approaches. *Journal of Information Science*, 36(3):306–323, 2010. (Cited on page 62.)

René Witte, Ninus Khamis et Juergen Rilling : Flexible ontology population from text : The owlexporter. *In Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, 2010. (Cited on page 67.)

Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta et Tao Xie : Automated extraction of security policies from natural-language software documents. *In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12)*, pages 1–11, 2012. (Cited on page 66.)

Burcu Yildiz et Silvia Miksch : ontox - a method for ontology-driven information extraction. *In Proceedings of the 2007 international conference on Computational science and its applications (ICCSA'07) - Volume Part III*, pages 660–673, 2007. (Cited on pages 62 et 63.)

Xiaojin Zhu : Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, 2:3, 2006. (Cited on page 64.)