Bases de connaissance spatiales pour les outils d'aide à la conception architecturale

Vonlanthen, Mathieu

UNIVERSITÉ
DE GENÈVE
FACULTÉ DES SCIENCES
ÉCONOMIQUES ET SOCIALES

Bases de Connaissance Spatiales pour les Outils d'Aide à la Conception Architecturale

THÈSE

présentée à la Faculté des Sciences Economiques et Sociales
de l'Université de Genève

par
Mathieu Vonlanthen

sous la direction de
Prof. Gilles Falquet et
Prof. Pierre Pellegrino

pour l'obtention du grade de
Docteur ès sciences économiques et sociales
mention systèmes d'information

Membres du jury de thèse :

Prof. Dimitri Konstantas, Président du Jury (Université de Genève)
Prof. Gilles Falquet, Directeur de Thèse (Université de Genève)
Prof. Pierre Pellegrino, Directeur de Thèse (Université de Genève)
Prof. Djamal Benslimane, Membre du Jury (Université Claude Bernard, Lyon)
Dr. Laurent Moccozet, Membre du Jury (Université de Genève)

Thèse N° 775
Genève, le 27 mars 2012

La Faculté des sciences économiques et sociales, sur préavis du jury, a autorisé l'impression de la présente thèse, sans entendre, par là, n'émettre aucune opinion sur les propositions qui s'y trouvent énoncées et qui n'engagent que la responsabilité de leur auteur.

Genève, le 27 mars 2012

Le doyen
Bernard MORARD

Impression d'après le manuscrit de l'auteur

« 2.033 – La forme est la possibilité de la structure. »

Ludwig Wittgenstein,
Tractacus logico-philosophicus

Abstract

The central question addressed in this thesis is to determine how to formally model the knowledge used by architects in the design of a building. The resulting model needs to be integrated into both a computer-aided architectural design (CAAD) tools and a formal knowledge base. The goal is to provide a set of automatic reasoning mechanisms such as classification, deduction, or abduction to the architectural designer.

Whether implicitly or explicitly, CAAD tools support a specific design theory of architecture. The type of CAAD tools defined here is intended to make theses theories explicit through a knowledge base model. We have chosen an architectural design model based on the work of Emmanuelle P. Jeanneret. This model define abstract design elements such as grids, composition patterns and configurable architectural elements.

We have chosen the description logic (DL) as our main formalism. More specifically, we use $\mathcal{SROIQ}$ which is the underlying logic of the OWL2 semantic web standard. This logic is a subset of first order logic with decidable consistency and subsumption tasks. Description logic allows a terminological representation of architectural design elements. To take into account the spatial (topological or geometric) aspects we used specific spatial logics .

For each design element, a model combining the terminological and spatial domain has been defined. This model allows to apply consistently both classical reasoning algorithms in DL and algorithms specific to spatial logics such as RCC8.

A prototype was developed in order to test different formalisms, algorithms and reasoning methods described in this thesis. It consists of a composition patterns and architectural elements editor. Patterns and elements can be exported into a description logic representation. Different reasoners and logical tools can then be applied on this representation in order to verify the consistency of the project, classify project elements according to an ontology of architecture or match similar elements in the knowledge base. The designer can also apply rules in order to complete the project or obtain proposals for the creation of new elements through the mechanism of abduction. The results of theses reasoning tasks can then be imported back to the editor.

Résumé

La question centrale de cette thèse consiste à déterminer les moyens de modéliser de manière formelle les connaissances utilisées par les architectes dans la conception d'un édifice. Le modèle utilisé doit pouvoir être intégré dans un système associant les outils d'aide à la conception architecturale (CAAD) et les bases de connaissance formelles. Des raisonnements automatiques comme la classification, la déduction ou l'abduction peuvent ainsi être appliqués par le concepteur sur des éléments de projets présents dans la base de connaissance.

Que cela soit de manière implicite ou explicite, les outils d'aide à la conception architecturale sous-tendent une certaine manière de concevoir l'architecture. Le type d'outils CAAD défini ici, a pour but d'expliciter par le biais d'une base de connaissance le modèle de conception architecturale sous-jacent. Nous avons choisi, en nous basant sur les travaux d'Emmanuelle P. Jeanneret, d'utiliser un modèle de conception qui explicite les éléments de conception suivants : les trames, les schémas de composition et les éléments d'architecture paramétrables.

Le formalisme pivot choisi est celui de la logique descriptive (LD), plus particulièrement la logique $\mathcal{SROIQ}$ utilisée dans le standard OWL2. Cette logique est un sous-ensemble de la logique du premier ordre pour lequel les tests de consistance et de subsomption sont décidables. La logique descriptive $\mathcal{SROIQ}$ permet une représentation terminologique des éléments de conception architecturale. Pour prendre en compte la dimension spatiale, qu'elle soit de type topologique ou géométrique, il faut néanmoins lui adjoindre des logiques spécialisées dans la représentation de l'espace.

Pour chacun des éléments de conception, un modèle associant le domaine terminologique et le domaine spatial a été défini. Les modèles ainsi définis permettent à la fois d'appliquer de manière consistante les algorithmes de raisonnement classique en LD et les algorithmes spécifiques aux logiques spatiales telles que RCC8.

Le prototype développé a servi à tester les différents formalismes, algorithmes et méthodes de raisonnements décrits dans cette thèse. Il est con-

stitué d'un éditeur qui permet de créer des schémas de composition et de les habiller avec des éléments d'architecture. Le schéma et les éléments d'architecture peuvent ensuite être exportés dans une représentation en logique descriptive. Le concepteur peut ainsi accéder par le biais de la base de connaissance à des éléments de projets similaires ou répondant à des critères donnés. Les différents raisonneurs et outils logiques peuvent ensuite être appliqués sur cette représentation de manière à vérifier la consistance du projet ou bien à classifier des éléments de projet selon une ontologie de l'architecture. Le concepteur peut aussi compléter automatiquement sont projet par l'application de règles ou bien obtenir des propositions de création d'éléments nouveau par le biais du mécanisme d'abduction. Les résultats peuvent ensuite être réimportés et modifiés, à nouveau, par le biais de l'éditeur.

Remerciements

Je tiens ici à remercier Gilles Falquet qui m'a donné l'envie de me lancer dans cette aventure et avec qui j'ai eu de nombreuses discussions fructueuses, Pierre Pellegrino pour m'avoir fait comprendre que l'architecture ne traite pas seulement des bâtiments, Emmanuelle P. Jeanneret pour m'avoir transmis sa vision de l'architecture, Daniel Coray pour son analogie entre TEX et la conception architecturale, Krishnendra Shekawat pour ses algorithmes de composition spatiale.

Je tiens à remercier Nizar Ghoula, Hélène De Ribaupierre, Camille Tardy, Alain Junger, El Mustapha El Atifi, Claire-Lise Mottaz, Saïd Radhouani et Jean-Claude Ziswiler pour avoir partagé avec moi la vie de thésard, Claudine Métral pour sa franchise, Jean-Pierre Hurny pour ses dicussions sur les neutrinos.

Je tiens à remercier Dimitri Konstantas pour avoir accepté de présider mon jury, Laurent Moccozet pour l'intérêt qu'il a porté à mon travail et Djamal Benslimane pour ses suggestions avisées concernant les services web.

Je tiens finalement à remercier Alice ma compagne pour son aide et sa relecture attentive.

Table des matières

Chapitre 1

Introduction

Cette thèse s'inscrit dans le cadre du projet interdisciplinaire « Formalisation et sens du projet architectural », financé par le Fonds National Suisse de la Recherche Scientifique (FNS). Ce projet se situe à la jonction des sciences de l'architecture, des sciences de l'informatique et des mathématiques.

Depuis plusieurs décennies déjà, l'omniprésence des technologies de l'information a influencé la manière de concevoir les édifices. L'augmentation massive des capacités de calcul permet de réaliser des bâtiments qui n'auraient jamais pu voir le jour autrement. Les outils d'aide à la conception architecturale[95] (CAAD) n'ont cependant jamais encore vraiment intégré les dernière avancées dans le domaine de la représentation formelle des connaissances.

Aujourd'hui, les outils d'aide à la conception architecturale, comme par exemple Autocad ou Archicad, sont principalement utilisés comme des outils de dessin assisté par ordinateur. Ils permettent d'effectuer d'autres tâches, comme par exemple des transformations géométriques complexes, des simulations acoustiques et thermiques, ou bien l'annotation des différentes parties d'un édifice avec une bibliothèque d'éléments standard, comme par exemple l'*Industry Foundation Classes* (IFC)[32]. Cependant, ces outils ne permettent pas d'accéder à des bases de connaissance contenant des éléments de projets antérieurs ou des principes généraux concernant l'architecture. Ils ne permettent pas de formaliser des connaissances abstraites, ni d'appliquer des procédures de raisonnement automatiques.

1.1 Contexte scientifique

Tim Berners-Lee, le concepteur du World Wide Web, a lancé un vaste programme de recherche par le biais de son article, paru dans le journal

Scientific American, intitulé *The semantic web*[13]. L'idée derrière ce terme consiste à formaliser explicitement la sémantique des données échangées sur le web, de manière à permettre leur interopérabilité sans avoir à définir des standards spécifiques pour chaque type d'application. Les outils et les standards développés pour le web sémantique permettent donc une représentation explicite des connaissances. Ils permettent aussi d'effectuer des inférences à partir des connaissances actuelles. La déduction, l'induction et l'abduction sont les principales procédures d'inférence utilisées. Le formalisme logique choisi par l'organisme de standardisation W3C est la logique descriptive[10]. Cette logique a pour avantage d'avoir des algorithmes de raisonnement décidables et de disposer d'une représentation standard largement reconnue[86]. La logique descriptive peut être vue comme un sous-ensemble de la logique du premier ordre tendant à maximiser l'expressivité tout en maintenant la décidabilité. De nouveaux résultats allant dans ce sens sont régulièrement publiés[9]. En logique descriptive, on utilise des classes d'instances appelées concepts et des relations entre des classes d'instances appelées rôles. La procédure de raisonnement standard est la subsomption, qui permet de déterminer si un concept est plus, ou moins général qu'un autre. Les autres procédures de raisonnement peuvent être ramenées à la subsomption, comme par exemple la procédure de vérification d'instances, qui permet à partir d'une instance donnée de déterminer les concepts auxquels elle appartient. L'abduction[54], qui détermine les hypothèses nécessaires pour qu'une instance appartienne à un concept, est un mécanisme d'inférence également disponible en logique descriptive. L'utilisation de règles[48] permet quant à elle de détecter une situation donnée et de déclencher une action correspondante.
L'application de la logique aux outils d'aide à la conception architecturale a été étudiée notamment par Mitchell[66] et par Gero[34]. Les formalismes utilisés dans ces travaux sont principalement la logique du premier ordre et les grammaires de formes. Les procédures de raisonnement issues de la logique descriptive ont déjà été appliquées aux outils CAAD[21][55][104].

1.2 Problématique générale

Aujourd'hui, les outils de conception architecturale[95], comme par exemple Autocad ou Archicad, ont comme vocation principale la représentation de bâtiments. Ils permettent d'effectuer des transformations géométriques complexes, de simuler l'acoustique, la thermique, la diffusion de la lumière ainsi que d'annoter les différentes parties d'un bâtiment avec une bibliothèques d'éléments standard (IFC)[32]. Cependant ces outils n'offrent pas la possibilité de travailler sur les concepts et les idées qui sont à la base d'un

projet architectural. Ils ne permettent pas de formaliser des connaissances abstraites, ni d'appliquer des procédures de raisonnement automatiques.

La question centrale de cette thèse consiste à déterminer les moyens permettant de formaliser les connaissances utilisées par les architectes dans la conception d'un édifice. Cette formalisation devra permettre d'associer des éléments logiques et des éléments géométriques, dans le but d'appliquer des procédures de raisonnement automatique sur les éléments abstraits qui permettent à un architecte de concevoir un édifice. La formalisation devra permettre, notamment, de vérifier automatiquement des contraintes, de réutiliser des éléments de projets venant d'une base de connaissance, de classifier des éléments de projets par rapport à une base de connaissance ou d'appliquer des règles de construction. La problématique consiste plus spécifiquement à déterminer en quoi les derniers développements de la logique descriptive et de la logique de l'espace permettent d'assister un architecte dans son travail de conception d'un édifice. L'idée est de modéliser les éléments abstraits qui permettent de concevoir les édifices, comme par exemple les trames ou les schémas de composition, afin de les lier aux outils d'aide à la conception architecturale déjà existants (CAAD)[53]. Ces outils devraient ainsi pouvoir profiter des avancées issues des efforts de recherche liés au web sémantique que ce soit dans le domaine des bases de connaissance[37][41] ou dans le domaine du raisonnement automatique[89][31][43].

Cette problématique générale induit un certain nombre de questions non triviales. Principalement, il s'agira de déterminer un ensemble de formalismes, permettant d'une part d'exprimer les problématiques propres à la conception architecturale, et d'autre part d'appliquer des procédures de raisonnement calculables. Il s'agira aussi d'étudier les moyens permettant de connecter ces différents formalismes entre eux.

Il s'agira, en outre, de décrire un système d'aide à la conception architecturale qui intègre par le biais d'une base de connaissance de l'architecture, et ce de manière explicite, des concepts clés de la théorie de l'architecture. Les différentes parties de ce système sont liées à des formalismes logiques et à des algorithmes pour la plupart déjà existants. La théorie de l'architecture que nous avons utilisée se base sur des éléments de conception abstraits, comme par exemple des trames ou des schémas de composition, ainsi que sur des éléments concrets, comme par exemple des éléments d'architecture paramétrables. Ces éléments sont modélisés puis stockés dans la base de connaissance. La base de connaissance est constituée d'une partie terminologique, qui définit des classes d'instances ayant des propriétés communes, et d'une partie assertionelle qui définit les instances et leurs relations. Cette base de connaissance contient d'une part des éléments spécifiques à des styles et à des architectes reconnus, et d'autre part des éléments de projets antérieurs.

Le modèle est basé sur un formalisme logique permettant d'appliquer des procédures de raisonnement automatique.

1.3 Démarche proposée

Dans un premier temps, nous allons, avec l'aide des théoriciens de l'architecture et plus particulièrement des travaux de Emmanuelle P. Jeanneret[52], déterminer des éléments abstraits de conception architecturale, issus des travaux des grands maîtres de l'architecture, comme par exemple Andrea Palladio, Louis Kahn ou Le Corbusier. Nous utiliserons des formalismes issus des derniers développements de la logique descriptive[10] et de la logique de l'espace[44][57]. Avec l'aide des architectes travaillant sur le projet FNS, Emmanuelle P. Jeanneret et Pierre Pellgrino, les concepts et la terminologie propres à l'architecture seront encodés dans une base de connaissance. Les techniques classiques généralement utilisées par les architectes pour élaborer des plans, comme par exemple les trames ou les schémas de composition, seront modélisés. D'autres techniques utilisées dans les CAAD comme les éléments d'architecture paramétrables seront elles aussi modélisées. Ces différents éléments seront organisés et mis en relation dans une base de connaissance de l'architecture.

Les représentations spatiales et terminologiques seront liées entre elles, par exemple par le formalisme des $\mathcal{E}$-connections[59]. Les représentations logiques seront associées à des représentations issues de la programmation orientée objet.

Les procédures de raisonnement mentionnées plus haut pourront être utilisées pour : vérifier la consistance d'un projet ; vérifier des contraintes et des objectifs spécifiés par un architecte ; obtenir une assistance permettant la résolution de contradictions entre les objectifs et les contraintes ; réutiliser des éléments de projets antérieurs ; classifier des éléments de projets dans une base de connaissance de l'architecture ; appliquer des règles de construction automatiques ; obtenir la liste des actions nécessaires pour atteindre un objectif donné ; inférer des nouvelles solutions partielles donnant au concepteur des propositions de modifications qu'il n'aurait pas forcément découvertes seul. Les moteurs de raisonnement automatiques, tel que *Pellet*[89], *Jess*[31] ou *Racer*[43], permettront par le biais de la subsomption, de l'abduction et de l'application de règles de réaliser les différentes procédures de raisonnements. L'interaction avec des algorithmes externes aux formalismes logiques, comme par exemple les algorithmes de traitement de graphes ou les algorithmes de satisfaction de contraintes, sera aussi étudiée.

Un prototype permettant l'interaction d'éléments géométriques et d'éléments logiques sera implémenté. Il permettra d'appliquer des procédures de raisonnement automatiques à des problèmes architecturaux. Le processus de conception sera vu comme l'interaction d'un architecte humain et d'une machine, tendant à une solution satisfaisante pour l'architecte.

Les éléments stockés dans la base de connaissance seront ainsi utilisés en combinaison avec les éléments correspondant au projet en cours de conception. De nouvelles classes d'éléments pourront être construites en combinant des termes décrivant des concepts généraux de l'architecture, et des termes décrivant des concepts spécifiques à un style ou à un projet donné. Les éléments de la base de connaissance qui sont les plus semblables à un élément donné d'un projet pourront aussi être retrouvés. Enfin, et c'est un des points importants de ces systèmes, des raisonnements de type déductif et abductif pourront être appliqués sur les éléments du projet, ainsi que sur les éléments de théorie générale présents dans la base de connaissance. L'architecte, en collaboration avec ces raisonnements automatiques, bénéficiera des connaissances accumulées antérieurement. Il pourra ainsi, tout au long de la phase de conception, interagir soit de manière directe, soit par le biais de raisonnements automatiques avec la base de connaissance.

Chapitre 2

Formalismes utilisés

Dans ce chapitre, nous allons décrire les formalismes dont nous aurons besoin par la suite pour modéliser les éléments de conception architecturale. Dans un premier temps, nous allons traiter des ontologies, nécessaires à la structuration de la base de connaissance. La syntaxe, la sémantique et les propriétés de la logique descriptive, utilisée comme formalisme de référence, seront ensuite décrits. Les règles logiques et les logiques spatiales topologiques et métriques seront ensuite étudiée.

2.1 Ontologies

2.1.1 Introduction

Le terme d'ontologie, avant d'être largement utilisé dans les domaines de la gestion de la connaissance et de l'intelligence artificielle, était déjà utilisé par les Grecs anciens pour dénommer la science de l'être. On peut considérer que la première ontologie, au sens de la gestion de la connaissance, a été conçue par les pré-socratiques. Elle divisait la totalité de l'être en quatre éléments fondamentaux : le feu, l'eau, l'air et la terre. Un des grands problèmes philosophiques était, et est toujours, la dualité entre l'essence et l'existence. Héraclite explicitait cette dualité avec sa célèbre phrase « On ne peut pas se baigner deux fois dans le même fleuve. »En effet, l'eau dans un fleuve n'est jamais la même, et pourtant le fleuve est toujours là. Platon quant à lui pensait que le *logos*, le monde des idées, était la source même de l'existence.

Ces considérations philosophiques permettent déjà de déterminer que les ontologies informatiques devront traiter de l'essence des objets considérés. Ce sont donc les concepts ou classes qui seront représentés dans les ontologies

informatiques plutôt que les individus ou objets.

2.1.2 Définition des ontologies

Le terme ontologie, tel qu'il est utilisé dans le domaine de la gestion de la connaissance, a été repris de la philosophie où il signifie : explication systématique de ce qui existe. Il a été proposé d'utiliser Ontologie avec un 'O' majuscule pour le sens philosophique du terme et ontologie avec un 'o' minuscule pour le sens utilisé dans la gestion de la connaissance. Il existe un grand nombre de définitions du terme ontologie. Nous allons en explorer quelque unes. La définition la plus connue et la plus citée est celle de Gruber[41] : « Une ontologie est une spécification explicite d'une conceptualisation ». Quelques années plus tard, Borst[17] étend la définition de Gruber de la manière suivante : « Les ontologies sont définies comme une spécification formelle d'une conceptualisation partagée. »Borst exclut donc de sa définition les ontologies dont les concepts sont définis en langage naturel. De plus, par le terme *partagée*, il explicite le fait qu'une ontologie est l'émanation d'une communauté de pensée et non pas d'un individu isolé. La définition de Uschold et al.[100] donne une définition d'une ontologie basée sur sa structure : « Une ontologie peut prendre différentes formes, mais elle inclura nécessairement un vocabulaire des termes, et une certaine spécification de leur signification. Cela inclut des définitions et des indications sur comment les concepts sont reliés, imposant collectivement une structure sur le domaine et contraignant les interprétations possibles des termes. »

2.1.3 Distinction sur les types d'ontologies

Il existe, comme nous le verrons, un grand nombre de distinctions que nous pouvons faire sur les différents types d'ontologies[37]. La distinction la plus utile et la plus courante est la distinction entre ontologie légère et ontologie lourde. Une ontologie légère est une ontologie qui contient une taxonomie, des relations entre concepts, et des propriétés de concepts, tandis qu'une ontologie lourde contient en plus des axiomes et des contraintes logiques qui restreignent l'interprétation des termes de l'ontologie.

Un autre type de distinction est fait sur le degré de formalisme. Une ontologie est hautement informelle si ses concepts sont exprimés en langage naturel. Elle est semi-informelle si ses concepts sont définis à partir d'une forme restreinte et structurée d'un langage naturel. Elle est semi-formelle si ses concepts sont définis par un langage artificiel et formellement défini. Finalement, elle est rigoureusement formelle si le langage permet de prouver

des propriétés comme la complétude et la consistance. Une ontologie est complète si et seulement si elle permet de déduire toutes les conclusions valides possibles et elle est consistante si et seulement si elle ne permet pas de déduire des conclusions invalides.

Les ontologies peuvent aussi être distinguées selon leur degré de formalisme[60] :

- Les vocabulaires contrôlés sont simplement une liste de termes appartenant à un domaine donné.
- Les glossaires sont une liste de termes, appartenant à un domaine donné, munis d'une définition en langage naturel.
- Les thésaurus sont des glossaires munis de relations entre les termes comme la synonymie ou l'antonymie, ils ne forment toutefois pas de hiérarchie explicite.
- Les hiérarchies informelles relient entre eux les termes que l'on peut classer dans une même catégorie, sans forcément respecter la sémantique du lien *est-un*, au contraire des hiérarchies formelles. Par exemple, dans un annuaire professionnel, sous l'entrée *enseignement*, peut être placé le terme *université* ainsi que le terme *professeur*.
- Les hiérarchies formelles respectent la sémantique du lien *est-un*, c'est-à-dire que si B *est-un* A, toutes les instances de la classe B sont aussi des instances de la classe A.
- Les ontologies légères contiennent une taxonomie, des relations entre concepts et des propriétés.
- Les ontologies lourdes contiennent en plus des axiomes et des contraintes logiques qui restreignent l'interprétation des termes de l'ontologie.

Enfin, les ontologies peuvent être distinguées par le sujet de leur conceptualisation :

- Les ontologies de représentation de la connaissance[101] modélisent les primitives de représentation de la connaissance selon un paradigme donné. Il s'agit donc de méta-modèle comme par exemple une ontologie du langage OWL encodée elle-même en OWL.
- Les ontologies générales[101][67] sont utilisées pour représenter des connaissances valables pour un grand nombre de domaines différents. Ces ontologies peuvent modéliser le temps, l'espace, la causalité, etc.
- Les ontologies de haut-niveau modélisent le monde en partant du concept le plus général possible, quelquefois appelé *Thing* ou *Top*. Un exemple est l'ontologie standard de haut-niveau appelée SUO (Standard Upper Ontology)[70] définie par un groupe de l'IEEE.
- Les ontologies de domaine[67][101] modélisent un domaine spécifique

(biologie, architecture, mathématique,...) et se veulent réutilisables. Elles traitent des théories, principes et activités du domaine.
- Les ontologies de tâches[67][42] décrivent les concepts associés à l'exécution d'une tâche ou d'une activité indépendamment d'un domaine, par exemple le diagnostique, la planification ou la vente.
- Les ontologies de tâche-domaine sont des ontologies de tâche spécifiques à un domaine donné.
- Les ontologies de méthode décrivent les concepts et relations utilisées pour spécifier un processus de raisonnement ou pour accomplir une tâche donnée[99].
- Les ontologies d'application[101] modélisent les connaissances nécessaires à une application particulière.

2.1.4 Formalisme pour les ontologies

Les ontologies les plus simples comme les vocabulaires contrôlés ou les glossaires ne nécessitent pas de formalisme particulier. Les thésaurus peuvent facilement être modélisés en utilisant des graphes orientés et étiquetés. A partir du moment où l'on veut décrire précisément des classes d'instances et des relations, il faut utiliser des descriptions plus formelles. Gruber a proposé un modèle[41] utilisant la logique du premier ordre et les *frames* basé sur les langages Ontolingua et KIF[33]. Le problème rencontré lors de l'utilisation de la logique du premier ordre est la présence d'ensembles de prédicats non décidables. C'est pour résoudre ce problème que la logique descriptive est apparue. Les premiers systèmes développés sur la base de cette logique sont KL-ONE[51], Classic[64] et LOOM[63]. Aujourd'hui, le langage de référence standardisé par le W3C est OWL, qui est le successeur de OIL et DAML+OIL[45][65]. Les graphes conceptuels sont aussi utilisés pour modéliser des ontologies.

2.2 Les formalismes logiques utilisés

Le développement d'un formalisme logique nécessite plusieurs étapes. Dans un premier temps, il faut analyser ce qui doit pouvoir être exprimé en fonction d'une application envisagée. Il faut ensuite définir la syntaxe et la sémantique. L'étude théorique de la complexité des algorithmes d'inférence est ensuite nécessaire avant d'envisager l'implémentation de ces algorithmes.

Un formalisme logique est un système formel qui définit de manière univoque une syntaxe et une sémantique. La communication entre humains peut avoir lieu en gardant un certain nombre d'ambiguïtés. Par contre, lever les

ambiguïtés est une chose indispensable si, comme dans notre cas, l'on veut stocker dans une base de connaissance de grandes quantités d'éléments de projet et d'éléments de style. Les langages formels peuvent facilement être dotés de la propriété de non-ambigüité. L'utilisation d'un formalisme logique pour représenter les différents éléments de conception architecturale permet de lever les ambiguïtés de leur expression. De cette façon, chaque expression d'un langage formel désigne un ensemble d'instances bien déterminées. Cependant les correspondances entre les éléments du langage formel et les représentations internes des humains sont toujours sujettes à ambiguïtés.

La formalisation permet aussi d'utiliser des méthodes de raisonnement automatique comme la déduction ou l'abduction. La puissance de calcul de l'ordinateur peut ainsi, non seulement être appliquée au calcul numérique, comme dans le cas de la simulation de résistance des matériaux, mais aussi au calcul symbolique. Par symboles, nous entendons unités syntaxiques qui, combinées entre elles selon des règles de grammaire, forment un langage formel. Ce langage formel est interprété de manière univoque par le biais d'une sémantique elle aussi formelle. Dans le cas de ces systèmes, la fonction d'interprétation détermine, à partir d'une expression qui combine des unités syntaxiques, un sous-ensemble de tout ce que l'on peut désigner. Par exemple, en utilisant la syntaxe de Manchester, l'expression "Porte and (estComposéDe value Chêne or estComposéDe value Sapin) and aComposant some (PoignéeBouton and aPrix some int[<=60])" désigne toutes les portes en chêne ou en sapin qui possèdent un bouton de porte d'un prix inférieur à 60 francs. Cette expression désigne les portes qui sont connues, c'est-à-dire présentes dans la base de connaissance, ainsi que les portes "potentielles", qui ne sont pas présentes dans la base de connaissance.

Le formalisme que nous avons choisi pour la représentation terminologique est celui de la logique descriptive[10]. Cette logique a pour avantage d'avoir des algorithmes de raisonnement décidables et de disposer d'une représentation standard largement reconnue[86]. La logique descriptive peut être vue comme un sous-ensemble de la logique du premier ordre tendant à maximiser l'expressivité tout en maintenant la décidabilité. De nouveaux résultats allant dans ce sens sont régulièrement publiés[9]. En logique descriptive, on utilise des classes d'instances appelées concepts et des relations entre des classes d'instances appelées rôles. La procédure de raisonnement standard est la subsomption, qui permet de déterminer si une classe est plus, ou moins générale qu'une autre. Les autres procédures de raisonnement peuvent être ramenées à la subsomption, comme par exemple la procédure de vérification d'instances, qui permet, à partir d'une instance donnée, de déterminer les concepts auxquels elle appartient. L'abduction[54], qui détermine les hypothèses nécessaires pour qu'une instance appartienne à un concept, est un

mécanisme d'inférence également disponible en logique descriptive. L'utilisation de règles[48] permet quant à elle de détecter une situation donnée et de déclencher une action correspondante.

Comme exemple, on peut décrire l'instance *pf1* qui appartient au concept *Porte* et au concept *Fenêtre*. Cette instance est liée par la relation *aLargeur* à la valeur 3.4, elle est aussi liée par la relation *estConnexeA* à l'instance *m1* qui elle-même appartient au concept *Mur*. On décrit ainsi un graphe qui relie les différentes instances. Les concepts sont définis comme des combinaisons d'autres concepts qui regroupent des ensembles d'instances. Par exemple, le concept *PorteFenetre* est construit comme l'intersection du concept *Porte* et du concept *Fenêtre*.

Un formalisme traitant de l'architecture doit nécessairement pouvoir exprimer ce qui a trait à l'espace. L'espace peut être représenté soit d'un point de vue métrique en définissant des distances, des longueurs, des hauteurs, soit d'un point de vue topologique en définissant des relations entre des régions de l'espace.

Dans le cas d'une logique métrique, chaque instance spatiale est liée par une distance à chacune des autres instances spatiales. Tant les instances de type spatial que les instances de type terminologique sont classées dans l'ontologie. On peut ainsi définir des classes d'instances satisfaisant à la fois à des axiomes de distances et à des axiomes terminologiques. En utilisant la logique métrique dans le cas de l'architecture, on peut ainsi, par exemple, définir la classe de toutes les colonnes corinthiennes qui sont à une distance de moins de 10 mètres de l'autel et de plus de 4 mètres des marches du temple.

Dans le cas d'une logique topologique, chaque instance spatiale est associée à une région du plan ou de l'espace tridimensionnel. Les relations entre les régions spatiales ne sont pas forcément connues. En prenant cinq régions d'une usine correspondant à des zones fonctionnelles différentes, seules certaines contraintes peuvent être spécifiées. Les autres relations sont laissées ouvertes permettant ainsi d'obtenir un ensemble de configurations différentes mais toutes admissibles. Si l'on spécifie que chaque zone froide doit être reliée à au moins trois zones chaudes par un couloir, les relations spatiales entre les zones froides et les zones chaudes peuvent être soit *estConnectéA* soit *estNonConnectéA*.

On parle de modèle spatio-terminologique quand, dans un même formalisme, il est possible de voir un même élément soit du point de vue terminologique (c'est-à-dire sa description en termes de propriétés et de relations), soit d'un point de vue spatial (c'est-à-dire ses relations topologiques et métriques), et quand il est possible de construire des expressions qui tiennent

compte à la fois de l'aspect terminologique et de l'aspect spatial.

2.3 Logiques descriptives

2.3.1 Syntaxe et sémantique

La logique descriptive[10] est un formalisme basé sur les réseaux sémantiques de Quillian[80] qui sont des graphes orientés auxquels on associe des concepts aux nœuds et des relations aux arcs, et sur la sémantique des cadres de Minsky[1] où les concepts sont représentés par des frames auxquels sont associés des attributs, appelés slots.

En logique descriptive, les concepts sont des définitions intensives représentant un ensemble d'individus. Les rôles décrivent des relations binaires entre les concepts, ils permettent entre autre d'associer des propriétés aux concepts. Les individus sont des éléments qui peuvent appartenir à plusieurs concepts ou à aucun.

Une théorie en logique descriptive est composée d'une TBox et d'une ABox. La TBox pour *terminological box* contient des déclarations concernant les concepts et rôles de la théorie. La ABox pour *assertional box* contient des déclarations sur les individus de la théorie.

Il existe toute une famille de logique descriptive allant des logiques les moins expressives au plus expressives.

Au niveau syntaxique, deux types de symboles sont définis : les concepts atomiques notés $A, B, ...$ correspondant à des prédicats unaires et les rôles atomiques notés $R, S, ...$ qui correspondent quant à eux à des prédicats binaires. Les termes sont construits à partir des symboles de base en utilisant différents types de constructeurs, comme par exemple l'intersection de concepts notée $\sqcap$ qui représente l'ensemble des individus appartenant aux concepts C et D. Comme nous pouvons le constater, les expressions de logique descriptive sont sans variables, alors qu'en logique du premier ordre cela s'écrirait $C(x) \wedge D(x)$. Pour établir des relations entre les concepts, il existe un constructeur appelé restriction de valeur et noté $\forall$ qui représente l'ensemble des individus qui sont dans une relation R avec des individus appartenant au concept C. Par exemple $\forall hasChild.Girl$ représente tous les individus qui n'ont que des filles. Les symboles et constructeurs que nous venons de décrire sont suffisants pour le langage le plus simple de la logique descriptive, à savoir $\mathcal{FL}_0$; en ajoutant de nouveaux constructeurs on obtient

Constructeur	**Syntaxe**	**Langage**
Concept	A	
Rôle	R	
Intersection	$C \sqcap D$	
Restriction de valeur	$\forall R.C$	
Quantification existentielle limitée	$\exists R$	
Top	$\top$	
Bottom	$\bot$	
Négation atomique	$\neg A$	
Négation	$\neg C$	$\mathcal{C}$
Union	$C \sqcup D$	$\mathcal{U}$
Restriction existentielle	$\exists R.C$	$\mathcal{E}$
Restriction de nombre	$(\geq nR)(\leq nR)$	$\mathcal{N}$
Nominaux	$a_1, ..., a_n$	$\mathcal{O}$
Hiérarchie de rôle	$R \sqsubseteq S$	$\mathcal{H}$
Rôle inverse	R^-	$\mathcal{I}$
Restriction de nombre fonctionnelle	$(\geq nR)(\leq nR)$	$\mathcal{F}$
Restriction de nombre qualifiée	$(\geq nR.C)(\leq nR.C)$	$\mathcal{Q}$

TABLE 2.1 – Constructeurs de la logique descriptive

des langages de plus en plus expressifs mais aussi de plus en plus difficiles à traiter pour les moteurs d'inférences. Le tableau 2.1 donne les constructeurs les plus courants. Comme nous l'avons dit, l'utilisation d'intersection et de restriction de valeur avec des concepts et des rôles permet de construire le langage $\mathcal{FL}_0$; si on ajoute la quantification existentielle limitée ainsi que le concept Top ($\top$) contenant toutes les instances et le concept Bottom ($\bot$) ne contenant aucune instance, on obtient alors le langage $\mathcal{FL}^-$. En ajoutant encore la négation atomique, on obtient le langage $\mathcal{AL}$ qui permet de construire tous les autres langages en utilisant les symboles de la table, comme par exemple les langages $\mathcal{ALC}$ et $\mathcal{ALCUE}$ qui sont d'ailleurs équivalents puisque l'union et la restriction existentielle peuvent être exprimées par la négation. Il faut encore noter que les deux précédent langages sont parfois abrégés en $\mathcal{S}$ ainsi $\mathcal{SHIQ}$ est équivalent à $\mathcal{ALCHIQ}$.

Une sémantique formelle peut être définie en considérant une *interprétation* $\mathcal{I}$ consistant en un ensemble non-vide $\Delta^{\mathcal{I}}$ appelé domaine d'interprétation et une fonction d'interprétation, qui associe à chaque concept atomique A un ensemble $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ et à chaque rôle atomique R une relation binaire $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. La sémantique est étendue aux différents constructeurs par le tableau 2.2.

Constructeur	**Syntaxe**	**Interprétation**
Concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Rôle	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Restriction de valeur	$\forall R.C$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1, d_2) \rightarrow d_2 \in C^{\mathcal{I}})\}$
Quantification existentielle limitée	$\exists R$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1, d_2))\}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Négation atomique	$\neg A$	$A^{\mathcal{I}} \backslash \Delta^{\mathcal{I}}$
Négation	$\neg C$	$C^{\mathcal{I}} \backslash \Delta^{\mathcal{I}}$
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Restriction existentielle	$\exists R.C$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1, d_2) \wedge d_2 \in C^{\mathcal{I}})\}$
Restriction de nombre	$(\geq nR)(\leq nR)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \{d_2 \mid R^{\mathcal{I}}(d_1, d_2)\}\| \geq n\}$
Nominaux	$a_1, ..., a_n$	$\{d \in \Delta^{\mathcal{I}} \mid d = a_i^{\mathcal{I}}$ pour un $a_i\}$
Hiérarchie de rôle	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Rôle inverse	R^-	$\{(d_1, d_2) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d_2, d_1)\}$
Restriction de nombre qualifiée	$(\geq nR.C)(\leq nR.C)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \{d_2 \mid R^{\mathcal{I}}(d_1, d_2), d_2 \in C^{\mathcal{I}}\}\| \geq n\}$

TABLE 2.2 – Sémantique des constructeurs

Les axiomes terminologiques sont donnés sous la forme :

$$C \sqsubseteq D \qquad (R \sqsubseteq S) \qquad ou \qquad C \equiv D \qquad (R \equiv S) \tag{2.1}$$

Si $\mathcal{T}$ est un ensemble d'axiomes alors l'interprétation $\mathcal{I}$ satisfait $\mathcal{T}$ si elle satisfait chacun des axiomes de $\mathcal{T}$. Si $\mathcal{I}$ satisfait un axiome (resp. un ensemble d'axiomes) alors $\mathcal{I}$ est appelé *modèle* de cet axiome (resp. de cet ensemble d'axiomes). Deux axiomes ou deux ensembles d'axiomes sont *équivalents* si ils ont les mêmes modèles. Une égalité dont la partie de gauche est constituée d'un concept atomique est appelée *définition* et sert à définir des *noms symboliques*. Dans une TBox $\mathcal{T}$ chaque nom symbolique doit être défini au plus une fois. Dans une TBox $\mathcal{T}$ l'ensemble des concepts atomiques qui apparaissent à gauche des définitions est appelé *ensemble des symboles de nom* $\mathcal{N}_\mathcal{T}$ et l'ensemble des concepts qui apparaissent seulement à droite des définitions est appelé *ensemble des symboles de base* $\mathcal{B}_\mathcal{T}$. Une interprétation de base $\mathcal{J}$ pour $\mathcal{T}$ est une interprétation qui interprète seulement les symboles de base appartenant à $\mathcal{B}_\mathcal{T}$. Une interprétation $\mathcal{I}$ qui interprète aussi les symboles de noms appartenant à $\mathcal{N}_\mathcal{T}$ est une *extension* de $\mathcal{J}$ si elle a le même domaine que $\mathcal{J}$ c'est à dire si $\Delta^\mathcal{I} = \Delta^\mathcal{J}$. La TBox $\mathcal{T}$ est appelée *définitionnelle* si pour toute interprétation de base $\mathcal{J}$, il existe exactement une extension $\mathcal{I}$ qui est un modèle de $\mathcal{T}$. Il apparait comme évident que si une terminologie (TBox) est définitionnelle alors les terminologies équivalentes le sont aussi. On dit que le concept A *utilise directement* le concept B si B apparait dans la partie de droite d'une définition et A dans la partie de gauche. La relation *utilise* est définie comme la fermeture transitive de la relation *utilise directement*. $\mathcal{T}$ contient un cycle ssi il existe un concept atomique qui s'utilise lui-même.

$$\mathsf{Human} \equiv \mathsf{Animal} \sqcap \forall \mathsf{hasParent}.\mathsf{Human} \tag{2.2}$$

La définition 2.2 qui est cyclique est composée du symbole de nom Human et des symboles de base hasParent et Animal. Nous pouvons voir que, pour une interprétation de base $\mathcal{J}$, il n'existe pas d'extension unique. Par exemple en interprétant hasParent comme une relation entre un animal et ses parents, il existe plusieurs interprétations possibles de Human soit comme l'ensemble de tous les animaux soit comme n'importe quel ensemble d'animaux qui contient ses parents. Il existe donc des terminologies cycliques qui ne sont pas définitionnelles. A l'opposé, une terminologie acyclique est toujours définitionnelle. On peut s'en convaincre en remplaçant tous les symboles de nom qui apparaissent dans la partie de droite d'une définition par le concept que le symbole dénote. La terminologie ainsi obtenue est appelée l'*expansion*.

Theorem 2.3.1 *Soit $\mathcal{T}$ une terminologie acyclique et $\mathcal{T}'$ son expansion, alors*

(i) $\mathcal{T}$ *et* $\mathcal{T}'$ *ont les mêmes symboles de base et les même symboles de nom.*
(ii) $\mathcal{T}$ *et* $\mathcal{T}'$ *sont équivalentes.*
(iii) $\mathcal{T}$ *et* $\mathcal{T}'$ *sont définitionnelles.*

Nous allons voir qu'il existe des terminologies cycliques définitionnelles mais qu'elles peuvent être réduites en terminologies acycliques. Considérons l'axiome suivant :

$$A \equiv \forall R.B \sqcup \exists R.(A \sqcap \neg A) \tag{2.3}$$

qui est cyclique mais qui peut être réduit en :

$$A \equiv \forall R.B \tag{2.4}$$

Ceci est un exemple de la situation générale qui découle du théorème 2.3.2 qui est une reformulation du théorème de définissabilité de Beth.

Theorem 2.3.2 *Toute* $\mathcal{ALC}$*-terminologie définitionelle est équivalente à une terminologie acyclique.*

Selon la sémantique que nous avons étudiée jusqu'à présent, appelée *sémantique descriptive*, les terminologies sont définitionnelles seulement si elles sont essentiellement acycliques. Pour utiliser des définitions cycliques, il faut introduire les sémantiques de point fixe. Considérons l'exemple d'un homme qui n'a que des descendants masculins.

$$\mathsf{Momd} \equiv \mathsf{Man} \forall \mathsf{hasChild}.\mathsf{Momd} \tag{2.5}$$

L'utilisation d'un cycle dans la définition parait ici appropriée.
Définissons maintenant la sémantique de point fixe. Considérons l'application $\mathcal{T}$ qui associe à chaque symbole de nom A sa description $\mathcal{T}(A) = C$. Une interprétation $\mathcal{I}$ est un modèle de $\mathcal{T}$ ssi $A^{\mathcal{I}} = (\mathcal{T}(A))^{\mathcal{I}}$. On introduit ensuite une application de l'ensemble des extensions d'une interprétation de base $\mathcal{J}$ dans lui-même $\mathcal{T}_{\mathcal{J}} : Ext_{\mathcal{J}} \rightarrow Ext_{\mathcal{J}}$ tel que $A^{\mathcal{T}_{\mathcal{J}}(\mathcal{I})} = (\mathcal{T}(A))^{\mathcal{I}}$. Le résultat suivant permet de définir un modèle pour les terminologies cycliques.

Theorem 2.3.3 *Soit* $\mathcal{T}$ *une terminologie,* $\mathcal{I}$ *une interprétation, et* $\mathcal{J}$ *la restriction de* $\mathcal{I}$ *aux symboles de base de* $\mathcal{T}$*, alors* $\mathcal{I}$ *est un modèle de* $\mathcal{T}$ *ssi* $\mathcal{I}$ *est un point fixe de* $\mathcal{T}_{\mathcal{J}}$*.*

Dans ce cadre, une terminologie est *définitionnelle* ssi toute interprétation de base $\mathcal{J}$ a une extension unique qui est un point fixe de $\mathcal{T}_{\mathcal{J}}$.

Nous allons regarder un exemple qui permet de comprendre pourquoi les terminologies cycliques ne sont pas définitionnelles. Prenons la terminologie

$\mathcal{T}^{Momd}$ définie plus haut. Prenons une interprétation de base $\mathcal{J}$ définie comme suit :

$$\begin{aligned}
\Delta^{\mathcal{J}} &= \{Charles_1, Charles_2, ...\} \cup \{James_1, ..., James_{Last}\} \\
Man^{\mathcal{J}} &= \Delta^{\mathcal{J}} \\
hasChild^{\mathcal{J}} &= \{(Charles_i, Charles_{(i+1)} | i \geq 1\} \\
&\quad \cup \{(James_i, James_{(i+1)} | 1 \leq i < Last)\}
\end{aligned}$$

On remarque tout d'abord qu'un individu sans fils fait partie de Momd. $James_{Last}$ et par conséquent toute sa dynastie font donc partie de Momd. Il est facile de vérifier que l'extension $\mathcal{I}_1$ qui contient tous les James est un point fixe de $\mathcal{T}_{\mathcal{J}}$. Si un seul Charles est un Momd alors sa dynastie aussi, cela conduit à une deuxième extension $\mathcal{I}_2$ qui est aussi un point fixe de $\mathcal{T}_{\mathcal{J}}$.

Afin de donner un effet définitionnel à des terminologies cycliques, il faut définir un point fixe unique. A cet effet nous introduisons un ordre partiel $\preceq$ sur les extensions de $\mathcal{J}$. On dit que $\mathcal{I} \preceq \mathcal{I}'$ si $A^{\mathcal{I}} \subseteq A^{\mathcal{I}'}$ pour tout symbole de nom dans $\mathcal{T}$. Un point fixe $\mathcal{I}$ de $\mathcal{T}_{\mathcal{J}}$ est un point fixe minimum si $A^{\mathcal{I}} \subseteq A^{\mathcal{I}'}$ pour tout point fixe $\mathcal{I}'$. On définit de la même manière le point fixe maximum.

De manière à identifier les terminologies qui ont la propriété d'avoir pour toute interprétation de base un point fixe minimum et un point fixe maximum, nous allons utiliser un résultat de la théorie des treillis. Un treillis est complet si toute famille d'éléments possède une borne supérieure minimum. Sur $EXT_{\mathcal{J}}$ nous avons défini la relation d'ordre partiel $\preceq$. Pour une famille d'interprétation $(\mathcal{I})_{i \in I}$ dans $EXT_{\mathcal{J}}$ nous définissons $\mathcal{I}_0 = \bigsqcup_{i \in I} \mathcal{I}_i$ comme l'union terme à terme des $\mathcal{I}_i$, c'est-à-dire que pour tout symbole de nom A nous avons $A^{\mathcal{I}_0} = \bigcup_{i \in I} A^{\mathcal{I}_i}$. $\mathcal{I}_0$ est la borne supérieur minimum des $\mathcal{I}_i$ ce qui montre que $(Ext_{\mathcal{J}}, \preceq)$ est un treillis complet.

Une fonction $f : L \to L$ sur un treillis $(L, \preceq)$ est *monotone* si $f(x) \preceq f(y)$ quand $x \preceq y$. Le théorème du point fixe de Tarski nous dit que pour une fonction monotone sur un treillis complet, l'ensemble des points fixes est non vide et forme lui-même un treillis complet. En particulier le plus petit et le plus grand point fixe.

Nous définissons une terminologie $\mathcal{T}$ comme monotone si l'application $\mathcal{T}_{\mathcal{J}}$ est monotone pour toute les interprétations de base $\mathcal{J}$. Par le théorème de Tarski, de telles terminologies ont un plus grand et un plus petit point fixe. Afin de reconnaitre les terminologies monotones, il existe un critère syntaxique très simple. On appelle une terminologie *sans négation* si aucune

négation n'apparaît à l'intérieur. Par induction sur la profondeur de la description des concepts, on peut vérifier que toute terminologie sans négation $\mathcal{ALCN}$ est monotone.

Theorem 2.3.4 *Si $\mathcal{T}$ est une terminologie sans négation et $\mathcal{J}$ une interprétation de base, alors il existe des extensions de $\mathcal{J}$ qui sont respectivement le modèle de plus grand point fixe et le modèle de plus petit point fixe.*

Il est possible d'obtenir une version plus forte de ce théorème en utilisant le *graphe de dépendance* $G_{\mathcal{T}}$ de la TBox. Les nœuds de ce graphe sont les symboles A de noms. Si la TBox $\mathcal{T}$ possède un axiome de la forme $A \equiv C$, alors pour chaque occurrence du symbole de nom A' dans C, il y a un arc de A vers A' dans $G_{\mathcal{T}}$. L'arc est noté positif si A' apparait dans C sous un nombre paire de négation et négatif sinon. Nous pouvons alors formuler le théorème suivant :

Theorem 2.3.5 *Soit $\mathcal{T}$ une terminologie, si tout cycle de $G_{\mathcal{T}}$ contient un nombre paire d'arcs noté négatif alors $\mathcal{T}$ est monotone. $\mathcal{T}$ possède donc un plus grand et un plus petit point fixe.*

Une terminologie satisfaisant la condition du théorème 2.3.5 est dite *syntaxiquement monotone.*
Dans les raisonneurs récents, les TBox ne sont pas définitionnelles et décrivent simplement des contraintes sur les modèles. On utilise pour cela les *axiomes d'inclusion générale de concepts* (general concept inclusion, GCI) de la forme :

$$C \sqsubseteq D$$

où C et D sont des concepts qui peuvent être composés. Une TBox composée de GCI est appelée TBox généralisée. En utilisant les GCI comme des contraintes sur des modèles, il n'est plus nécessaire d'utiliser des contraintes syntaxiques comme l'unicité du membre de gauche, l'acyclicité ou la monotonicité.

2.3.2 Tâches de raisonnement

Il existe différentes tâches de raisonnement que l'on peut effectuer sur une TBox $\mathcal{T}$:

- **Satisfiabilité** Un concept C est satisfiable par rapport à $\mathcal{T}$ si il existe un modèle $\mathcal{I}$ tel que $C^{\mathcal{I}}$ est non vide. Dans ce cas, on dit que $\mathcal{I}$ est un modèle de C.
- **Subsomption** Un concept C subsume un concept D, par rapport à $\mathcal{T}$, si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$. Dans ce cas on écrit $C \sqsubseteq_{\mathcal{T}} D$ ou $\mathcal{T} \models C \sqsubseteq D$.

- **Equivalence** Deux concepts sont dit équivalents, par rapport à $\mathcal{T}$, si $C^{\mathcal{I}} = D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$. Dans ce cas on écrit $C \equiv_{\mathcal{T}} D$ ou $\mathcal{T} \models C \equiv D$.
- **Disjoint** Deux concepts sont dit disjoints, par rapport à $\mathcal{T}$, si $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.

Chacune des tâches de raisonnement précédentes peut être réduite à la subsomption de la manière suivante :

- C est insatisfiable $\Leftrightarrow$ C est subsumée par $\bot$.
- C et D sont équivalents $\Leftrightarrow$ C est subsumé par D et D est subsumé par C.
- C et D sont disjoints $\Leftrightarrow$ $C \sqcap D$ est insatisfiable.

Les tâches de raisonnement peuvent aussi être réduites à la tâche d'insatisfiabilité de la manière suivante :

- C est subsumé par $D \Leftrightarrow C \sqcap \neg D$ est insatisfiable.
- C et D sont équivalents $\Leftrightarrow$ $(C \sqcap \neg D)$ et $(\neg C \sqcap D)$ sont insatisfiable.
- C et D sont disjoints $\Leftrightarrow C \sqcap D$ est insatisfiable.

Au niveau de la ABox, la tâche de raisonnement principale est celle de décider de la *consistance d'une ABox par rapport à une TBox*. On dit qu'une ABox $\mathcal{A}$ est consistante par rapport à une TBox $\mathcal{T}$ s'il existe une interprétation qui soit à la fois un modèle de $\mathcal{A}$ et de $\mathcal{T}$.

Dans le cas d'une TBox acyclique, il est possible de réduire la vérification de la consistance d'une ABox par rapport à une TBox à la vérification de la consistance d'une ABox par rapport à une TBox vide. Pour ce faire, on remplace chaque assertion $C(a)$ dans la ABox par l'assertion contenant l'expansion du concept $C'(a)$. L'expansion C' du concept C est obtenue en remplaçant dans la définition de C tous les concepts définis par leurs définitions.

Les autres tâches de raisonnement sur une ABox sont :

- la *vérification d'instance* qui consiste à déterminer si l'instance a appartient au concept C ;
- la *recherche d'instance* qui consiste à déterminer toutes les instances de la ABox qui appartiennent au concept C ;
- la détermination du *concept le plus spécifique* qui consiste à déterminer pour une instance a le plus petit concept C d'un ensemble de concepts donnée tel que $\mathcal{A} \models C(a)$. Le concept C devant être minimal par rapport à la relation de subsomption $\sqsubseteq$.

La recherche des instances a appartenant au concept C peut-être effectuée en vérifiant pour chaque instance présente dans la ABox si elle appartient au concept C. La détermination du concept le plus spécifique pour une instance a peut être effectuée en vérifiant pour chaque concept présent dans la TBox s'il contient l'instance a.

La consistance de la ABox par rapport à une TBox peut être réduite à la

vérification d'instances en vérifiant si $\mathcal{A} \not\models_{\mathcal{T}} \bot\ (a)$ où a est un nom d'individu quelconque. On peut aussi réduire la vérification d'instance $\mathcal{A} \models_{\mathcal{T}} C(a)$ à la vérification de la non consistance de $\mathcal{A} \cup \{\neg C(a)\}$. La vérification de satifiabilité d'un concept C par rapport à $\mathcal{T}$ peut être réduite à la vérification de la consistance de $\{C(a)\}$ par rapport à $\mathcal{T}$.

2.3.3 Propriété de modèle en arbre fini

Une logique descriptive possède la *propriété de modèle en arbre fini* si chaque concept satisfiable C_0 possède un modèle fini en forme d'arbre. La racine de l'arbre appartient à C_0. La logique $\mathcal{ALCN}$ possède la propriété de modèle en arbre fini.
Cette propriété explique que nous ne pouvons pas détecter des cycles en utilisant la vérification d'instances. Soit la TBox $\mathcal{T} = \{Cycle \equiv A \sqcap \exists R.(A \sqcap \exists R.Cycle)\}$ et la ABox $\mathcal{A} = \{A(a_1), A(a_2), R(a_1, a_2), R(a_2, a_1)\}$. Nous voulons déterminer si $\mathcal{A} \models_{\mathcal{T}} Cycle(a_1)$. Nous pouvons déterminer $\mathcal{I}_1, \mathcal{I}_2$ deux modèles de $\mathcal{A}$, ayant chacun le même domaine d'interprétation $\Delta^{\mathcal{I}_1} = \Delta^{\mathcal{I}_2}$ tel que $a_1 \in Cycle^{\mathcal{I}_1}$ et $a_1 \notin Cycle^{\mathcal{I}_2}$.

2.3.4 La logique $\mathcal{SROIQ}$

La logique $\mathcal{SROIQ}$[47] est celle utilisée dans le standard OWL2[73][46]. Il s'agit d'une extension de la logique $\mathcal{SHOIN}$[49], qui elle était utilisée pour la première version d'OWL[86]. La logique $\mathcal{SROIQ}$ est issue de la logique $\mathcal{ALC}$ munie de la fermeture transitive sur les rôles simples ($\mathcal{S}$), de la hiérarchie sur les rôles ($\mathcal{R}$), des nominaux ($\mathcal{O}$), des rôles inverses ($\mathcal{I}$) et des restrictions qualifiées de nombres ($\mathcal{Q}$).
D'autres opérateurs sur les rôles sont présents dans $\mathcal{SROIQ}$ comme :

- la disjonction de rôles,
- les rôles irréflexifs et réflexifs,
- les assertions négatives de rôles,
- l'inclusion de rôles complexes,
- le rôle universel U,
- le constructeur Self.

2.4 Règles et logiques descriptives

L'utilisation de règles en logique descriptive permet d'une part d'étendre l'expressivité, et d'autre part de faciliter la définition de concepts et d'axiomes.

2.4.1 RDF

On définit RDF avec quatre ensembles disjoints : un ensemble d'URIs $\mathcal{V}_{uri}$, un ensemble d'identifiants de blank nodes $\mathcal{V}_{bnode}$, un ensemble de littéraux bien formés $\mathcal{V}_{lit}$, un ensemble de noms de variables $\mathcal{V}_{var}$. Un triple RDF est un tuple $(s, p, o) \in (\mathcal{V}_{uri} \cup \mathcal{V}_{bnode}) \times \mathcal{V}_{uri} \times (\mathcal{V}_{uri} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit})$.

2.4.2 SPARQL

Les éléments de base de SPARQL sont les Basic Graph Patterns (BGP). Un BGP est une ensemble de triples contenant des variables. Une solution d'un BGP par rapport à un graphe RDF source G est une application μ de l'ensemble des variables présentes dans la requête vers l'ensemble des termes RDF, telle qu'une substitution de variables dans le BGP détermine un sous-graphe de G. Les BGP peuvent être combinés avec des opérateurs de l'algèbre relationnel comme la projection (SELECT), la jointure à gauche (OPTIONAL), l'union (UNION) et les contraintes (CONSTRAIN).

2.4.3 SPARQL-DL

SPARQL-DL[90][56] fournit un sous-langage de SPARQL munis d'une sémantique OWL-DL. SPARQL-DL génère des sous-graphes conforment à OWL-DL. SPARQL-DL est plus expressif que les langages de requêtes par logique descriptive déjà existant. Il permet en effet d'effectuer des requêtes sur la TBox, la ABox et la RBox (hiérarchie des rôles) en même temps. Son implémentation se veut relativement aisée si l'on utilise un moteur de raisonnement DL. Les atomes SPARQL-DL sont de la forme :

- $Type(a, C)$
- $PropertyValue(a, p, v)$
- $SameAs(a, b), DifferentFrom(a, b)$
- $EquivalentClass(C1, C2)$
- $SubClassOf(C1, C2)$
- $DisjointWith(C1, C2)$
- $ComplementOf(C1, C2)$
- $EquivalentProperty(p1, p2)$
- $SubPropertyOf(p1, p2)$
- $InverseOf(p1, p2)$
- $ObjectProperty(p), DatatypeProperty(p)$
- $Functional(p)$
- $InverseFunctional(p)$
- $Transitive(p)$

– $Symmetric(p)$
– $Annotation(s, p_a, o)$

Des variables peuvent être utilisées pour les noms de concepts et de propriétés.

2.4.4 Datalog

Une signature $\langle \mathbf{I}, \mathbf{P}, \mathbf{V} \rangle$ pour Datalog consiste en un ensemble fini de noms d'individus $\mathbf{I}$, un ensemble fini de noms de prédicats $\mathbf{P}$, et un ensemble fini de noms de variables $\mathbf{V}$. Chacun de ces ensembles étant mutuellement disjoint. La fonction $ar : \mathbf{P} \to \mathbb{N}$ associe un entier naturel $ar(P)$ pour chaque prédicat $P \in \mathbf{P}$ qui définit l'arité de P. On définit :

– un terme Datalog comme un élément $t \in \mathbf{I} \cup \mathbf{V}$ à savoir un nom de variable ou d'individu ;
– en atome Datalog comme une formule de la forme $P(t_1, ..., t_n)$ où $t_1, ..., t_n$ sont des termes Datalog et $P \in \mathbf{P}$ un prédicat Datalog d'arité n ;
– une règle Datalog comme une formule de la forme $\forall x_1, ..., \forall x_m . B_1 \wedge ... \wedge B_k \to H$, où $B_1, ..., B_k$ sont des atomes Datalog ou le symbole $\top$, H est un atome Datalog ou le symbole $\bot$, et les variables $x_1, ..., x_m$ sont exactement les variables qui apparaissent dans ces atomes.

Une règle avec $k = 1$ et $B_1 = \top$ est appelée un fait, et une règle avec $H = \bot$ est appelée une contrainte. La prémisse d'une règle Datalog est appelée le corps de la règle tandis que la conclusion est appelée la tête. Un ensemble de règles Datalog est appelé un programme Datalog.

2.4.5 Requêtes conjonctives distinguées

Les requêtes conjonctives distinguées ou Distinguished Conjunctive Query (DCQ)[96] permettent d'interroger une ABox. Elles sont aussi utiles pour formuler des contraintes d'intégrité. En utilisant les mêmes symboles que ceux utilisés dans la section sur la logique descriptive, on définit une requête atomique q comme un axiome de ABox contenant des variables.

$$q \leftarrow C(x) \mid R(x, y) \mid \neg R(x, y) \mid x = y \mid x \neq y$$

On définit une requête conjonctive Q par

$$Q \leftarrow q \mid Q_1 \wedge Q_2$$

Une requête conjonctive distinguée est une requête conjonctive ne contenant que des variables distinguées. A savoir des variables qui peuvent être liées uniquement à des individus nommés. Une affectation est une application

$\sigma : N_V \to N_I$ de l'ensemble des variables vers l'ensemble des individus. Une base de connaissance $\mathcal{K}$ satisfait une *DCQ* Q, avec une affectation σ, noté par $\mathcal{K} \models^\sigma Q$ si

$$\begin{array}{lll} \mathcal{K} \models^\sigma q & \text{ssi} & \mathcal{K} \models \sigma(q) \\ \mathcal{K} \models^\sigma Q_1 \wedge Q_2 & \text{ssi} & \mathcal{K} \models^\sigma Q_1 \text{ et } \mathcal{K} \models^\sigma Q_2 \end{array}$$

Une réponse à une requête $\mathbf{A}(Q, \mathcal{K})$ est l'ensemble des affectations qui satisfont une base de connaissance $\mathcal{K}$, autrement dit $\mathbf{A}(Q, \mathcal{K}) \models \{\sigma \mid \mathcal{K} \models^\sigma Q\}$. Une requête est vraie s'il existe au moins une affectation satisfaisante et fausse sinon.

Pour pouvoir exprimer des contraintes, il est parfois nécessaire d'utiliser l'opérateur NAF (Negation As Failure) **not**. En ajoutant cet opérateur aux requêtes *DCQ*, on obtient des requêtes DCQ^{not} dont la syntaxe est définie par

$$Q \leftarrow q \mid Q_1 \wedge Q_2 \mid \mathbf{not}\ Q$$

et dont la sémantique est définie par

$$\mathcal{K} \models^\sigma \mathbf{not}\ Q \quad \text{ssi} \quad \nexists \sigma' \text{ tel que } \mathcal{K} \models^{\sigma'} \sigma(Q)$$

On sait que SPARQL est suffisant pour exprimer DCQ^{not}. Les requêtes conjonctives sont implémentées dans le logiciel Pellet Integrity Constraint Validator développé par Clark&Parsia.

2.4.6 SWRL

Datalog et la logique descriptive sont tous les deux un fragment de la logique du premier ordre (FOL). SWRL prend comme parti de combiner les deux. Une signature SWRL est une signature Datalog $\langle \mathbf{I}, \mathbf{P}, \mathbf{V} \rangle$, on définit en plus un ensemble de noms de concepts $\mathbf{A} \subseteq \mathbf{P}$, un ensemble de noms de rôles simples $\mathbf{N_s} \subseteq \mathbf{P}$ et un ensemble de noms de rôles non-simples $\mathbf{N_n} \subseteq \mathbf{P}$ telle que :

- $A \in \mathbf{A}$ implique que $ar(A) = 1$, et
- $R \in \mathbf{N_s} \cup \mathbf{N_n}$ implique que $ar(R) = 2$

Tous les problèmes de raisonnement standard sont, pour SWRL ,indécidables même en restreignant la logique descriptive sous-jacente. Il est donc impossible de construire un moteur d'inférence qui permettent de calculer toutes les conclusions à partir d'un ensemble de règles arbitraires. Il est cependant possible de dériver des conclusions qui sont certaines, donc de construire un moteur d'inférence qui soit consistant mais incomplet. Il est aussi possible de restreindre l'ensemble des règles permises et d'obtenir un moteur d'inférence consistant et complet.

2.4.7 Règles par l'approche basée sur les axiomes

Il existe plusieurs approches permettant d'utiliser des règles en logique descriptive. L'approche que nous allons présenter ici est appelée approche basée sur les axiomes[9]. Elle consiste à étendre la base de connaissance DL Σ avec un langage du premier ordre, sans fonctions, de signature $\mathcal{A}$. La signature de Σ étant un sous-ensemble de $\mathcal{A}$. Une règle classique se présente sous la forme :

$$h_1 \wedge ... \wedge h_l \leftarrow b_1 \wedge ... \wedge b_l \tag{2.6}$$

Il faut maintenant définir ce que sont : un terme, un atome, et un littéral. Un terme est une constante de $\mathcal{A}$ ou un symbole de variable. Si $t_1, ..., t_n$ sont des termes et R est un prédicat d'arité n, $R(t_1, ..., t_n)$ est un atome. Un atome ou sa négation $\neg R(t_1, ..., t_n)$ sont des littéraux. Dans le cas de la règle 2.6 $h_1, ..., h_l$ sont les littéraux de la tête de la règle et $b_1, ..., b_n$ sont les littéraux du corps de la règle. Le graphe de co-référence d'un ensemble de littéraux a pour nœuds les littéraux et les variables ; les arrêtes indiquant par leurs étiquettes la position dans un littéral. Un ensemble de littéraux possède une forme d'arbre si son graphe de co-référence ne possède pas de cycle.

Il est maintenant possible de définir une condition suffisante pour qu'un ensemble de règles soit décidable.
Si l'ensemble de règles contient uniquement des règles-DL tel que :
- *il n'y ait pas de rôles atomiques avec négation,*
- *l'ensemble des termes de la tête ait une forme d'arbre,*
- *l'ensemble des termes du corps ait une forme d'arbre,*

alors le problème de l'implication logique est NexpTime*-complet pour les langages $\mathcal{ALCQI}$, OWL-Lite et OWL-DL.*

Si nous ajoutons aux conditions précédentes l'interdiction de constantes dans les règles, alors le problème de l'implication logique est ExpTime *à condidtion que la logique descriptive utilisée pour la base de connaissance soit elle même* ExpTime *(p.ex : ALCQI ou OWL-Lite).*

Si l est un littéral alors l ou *not* l sont des littéraux-NAF, (Negation As Failure). On définit les atomes-DL, les littéraux-DL, les littéraux-NAF-DL comme, respectivement, des atomes, des littéraux, des littéraux-NAF, dont les prédicats appartiennent à la signature de la base de connaissance Σ. Une règle-DL est une règle qui contient uniquement des littéraux-DL. Les variables distinguées sont les variables d'une règle qui apparaissent à la fois dans la tête et dans le corps de la règle, à savoir $D(r) = vars(H(r)) \cap vars(B(r))$. Une règle est *safe* si toutes les variables dans la tête de la règle sont distin-

guées. Une règle DL-safe est une règle pour laquelle chaque variable apparaît dans le corps de la règle à l'intérieur d'un atome non DL (un littéral positif). Une règle atomique est une règle ne contenant qu'un seul littéral dans sa tête.

Si l'on utilise uniquement des règles DL-safe atomique alors le problème de l'implication logique est décidable pour toute DL sous-jacente décidable.[69][84]

Les règles DL-safe assurent que chaque variable est attachée à un individu explicitement défini dans la ABox. Par exemple, la règle suivante n'est pas DL-safe :

$$ElementStyleAB(x) \leftarrow Element(x) \wedge hasStyle(x, a) \wedge hasStyle(x, b)$$

En effet les variables x et y sont dans des atomes DL, mais n'apparaissent pas dans un atome du corps qui n'est pas déclaré hors de la base de connaissance. Cette règle peut être transformée dans une règle DL-safe en ajoutant des littéraux non-DL spéciaux $\mathcal{O}(x)$, $\mathcal{O}(a)$,$\mathcal{O}(b)$ au corps de la règle et en ajoutant les faits $\mathcal{O}(c)$ pour chaque individu c. Il faut aussi être conscient du fait que les règles DL-safe permettent uniquement l'utilisation de concepts atomiques, mais cette restriction peut être contournée. Pour chaque concept complexe C, on peut en effet ajouter un concept atomique A et l'axiome $C \sqsubseteq A$ dans la TBox puis utiliser le concept A dans la TBox.

2.4.8 Description Logic Programs (DLP)

La DLP[40] peut être vue comme l'intersection de la logique descriptive et du fragment de Horn de la programmation logique. Les règles DLP peuvent être encodées dans une base de connaissance DL tout en conservant leur sémantique. De manière équivalente, le sous-ensemble DHL de la logique descriptive peut être encodée sous forme de règles de programmation logique et bénéficier des avantages des moteurs de règles.
Une clause en logique du premier ordre à la forme :

$$L_1 \vee ... \vee L_k$$

où les L_i sont des littéraux de la forme (1) A ou (2) $\neg A$, où A est un atome. Dans le cas (1), le littéral est dit positif et dans le cas (2), il est dit négatif. Une clause est dite de Horn si elle possède au plus un littéral positif. Une clause de Horn est dite définie quand exactement un littéral est positif. Une clause de Horn définie, aussi appelée règle de Horn peut donc être écrite sous la forme :

$$H \leftarrow B_1 \wedge ... \wedge B_m$$

La règle de programmation logique qui est syntaxiquement équivalente à cette règle de Horn est dite LP-équivalente. Généralement ces règles sont dite Datalog, si elle ne contiennent pas de symbole d'égalité, pas de négations et si elles sont safe (toutes les variables de la tête apparaissent aussi dans le corps).
Il est maintenant possible de projeter un sous-ensemble de la logique descriptive sur des règles de Horn définies. A titre d'exemple, l'axiome $\top \sqsubseteq \forall P.C$ qui définit le rang d'un rôle se traduit par la règle :

$$C(y) \leftarrow P(x, y)$$

. La description logique de Horn (Description Horn Logic) est le sous-ensemble de la logique descriptive qui peut se traduire en règles de Horn définies. Un programme de logique descriptive (Description Logic Programm, DLP) est l'ensemble de règles LP-équivalentes à un ensemble d'axiomes DHL.

2.5 Abduction

2.5.1 Définition générale

L'abduction[54][15][20][71][22][29] permet de répondre à la question : « Quel ensemble d'axiomes dois-je rajouter à ma théorie pour qu'un axiome donné puisse être déduit ? ». De manière formelle, un problème abductif est donné par un couple $\langle \mathcal{K}, \phi \rangle$, où $\mathcal{K}$ est un ensemble de formules logiques (la base de connaisance) et ϕ une formule logique (la requête) avec comme condition que $\mathcal{K} \not\models \phi$. Une formule α est une solution d'un problème abductif $\langle \mathcal{K}, \phi \rangle$ ssi $\mathcal{K} \cup \{\alpha\} \models \phi$. Plusieurs propriétés sont définies pour une solution abductive. La solution α est :

- consistante ssi $\mathcal{K} \cup \alpha \not\models \bot$,
- pertinente ssi $\alpha \not\models \phi$,
- minimale ssi pour toute solution consistante et pertinente β, si $\alpha \models \beta$ alors $\beta \models \alpha$.

La programmation logique inductive[103] permet, quant à elle, à partir d'un ensemble d'assertions vraies, d'un ensemble d'assertions fausses et d'une base de connaissance de générer des hypothèses qui permettent de déduire l'ensemble des assertions vraies et aucunes des assertions fausses. Contrairement à l'abduction, le raisonnement se fait dans un monde clos qui est celui de l'ensemble des individus présents dans la base de connaissance. L'abduction, elle, permet de travailler sur un monde ouvert ainsi que de générer des nouveaux individus.

2.5.2 Abduction en logique descriptive

En logique descriptive, cinq types d'abduction peuvent être définis :

- **L'abduction de concept** A partir d'un concept C trouver un concept H tel que $\mathcal{K} \models H \sqsubseteq C$. Un cas particulier de ce type d'abduction est l'**abduction de concept conditionnelle** où le but est de trouver un concept H tel que $\mathcal{K} \models H \sqcap D \sqsubseteq C$, pour un concept fixé D.
- **L'abduction de TBox** A partir d'un GCI $C \sqsubseteq D$ trouver un autre GCI $E \sqsubseteq F$ tel que $\mathcal{K} \cup \{E \sqsubseteq F\} \models C \sqsubseteq D$.
- **L'abduction de ABox** A partir d'une assertion $\phi(a)$ trouver un ensemble d'assertions A tel que $\mathcal{K} \cup A \models \phi(a)$.
- **L'abduction de base de connaissance** A partir d'un axiome de ABox ou de TBox ϕ trouver un ensemble d'axiomes de ABox et/ou TBox A tel que $\mathcal{K} \cup A \models \phi$.

Une solution créative dans l'abduction est définie comme une solution utilisant au moins un nom de concept, rôle ou individu qui n'a pas été défini dans la base de connaissance, autrement dit un anonyme.

La solution à ces problèmes d'abduction peut être contrainte à être formulée dans une logique de description plus simple que celle utilisée par C ou D. Par exemple, en interdisant certains constructeurs.

Selon l'article[29], l'abduction en logique descriptive est « un domaine de recherche quasiment entièrement ouvert ». Cet article liste de manière non-exhaustive certaines tâches restant à accomplir comme :

- l'identification de restrictions syntaxiques pertinentes sur l'ensemble des solutions ;
- la définition d'une notion appropriée de minimalité ;
- l'identification de logiques descriptives admettant des techniques de raisonnement basées sur les tableaux qui permettent l'abduction et l'étude de leur complexité.

2.6 Logiques topologiques

La représentation qualitative de l'espace prend comme parti de représenter des relations entre des entités plutôt que des distances entre des points. C'est une représentation plus proche du raisonnement humain que la représentation métrique. Il est possible de représenter tant des relations topologiques que des relations d'orientations.

Pour comprendre le problème de la représentation qualitative de l'espace il est d'abord nécessaire de définir ce que sont les problèmes de satisfaction

de contraintes.

2.6.1 Problèmes de satisfaction de contraintes

Un problèmes de satisfaction de contraintes (CSP)[5] consiste à déterminer l'état d'un ensemble d'objets de manière à ce qu'ils satisfassent un ensemble de contraintes. De nombreux problèmes comme le placement de rectangles dans une surface, le placement de huit reines sur un échiquier tel qu'elles ne s'atteignent pas ou le coloriage d'une carte avec quatre couleurs peuvent se réduire à des CSP.

De manière formelle, on définit m variables appartenant à $\mathcal{V} = \{x_1, ..., x_m\}$ sur un domaine $\mathcal{D}$ et un ensemble de contraintes Θ. Chaque contrainte consiste en une relation n-aire $R_i \subseteq \mathcal{D}^n$ et un n-tuple de variables $\langle x_{i_1}, ..., x_{i_n} \rangle$. Pour les contraintes binaires, on utilise la notation infixe $x_1 R x_2$. Une instantiation partielle de $k \leq m$ variables est une fonction $f : \mathcal{V}_k \subseteq \mathcal{V} \rightarrow \mathcal{D}$. Une instantiation satisfait une contrainte $R(x_{i_1}, ..., x_{i_n})$ ssi $\langle f(x_{i_1}), ..., f(x_{i_n}) \rangle \in R_i$. Un problème de satisfaction de contraintes (CSP) consiste en un ensemble de variables $\mathcal{V}$ sur un domaine $\mathcal{D}$ et un ensemble de contraintes Θ. Un CSP est consistant s'il possède une solution.

Dans le cas de contraintes binaires le CSP peut être représenté par un réseau de contraintes, chaque nœud étant une variable et chaque arrête étant une relation. On utilise la notation R_{ij} pour noter la relation contraignant la paire de variables $\langle x_i, x_j \rangle$.

Il est maintenant utile de définir un algèbre de relations pour pouvoir englober le cas particulier de RCC8 dans un cadre théorique plus général. Un algèbre de relation consiste en un ensemble de relations binaires $\mathcal{R}$ clos par les opérations d'union ($\cup$), d'intersection ($\cap$), de composition ($\circ$), de complément ($\bar{\cdot}$) et de conversion ($\cdot^\smile$). L'opération de composition est définie comme $R \circ S \stackrel{def}{=} \{\langle x, y \rangle \mid \exists z : \langle x, z \rangle \in R \wedge \langle z, y \rangle \in S\}$ et celle de conversion comme $R^\smile \stackrel{def}{=} \{\langle x, y \rangle \mid \langle y, x \rangle \in R\}$. L'algèbre de relations doit aussi contenir la relation vide $\emptyset$, la relation universelle $\divideontimes$ et la relation identité Id.

La propriété JEPD (Jointly Exhaustive and Pairwise Disjoint.) pour un ensemble de relations $\mathcal{R} = \{R_1, ..., R_n\}$ est d'un intérêt particulier. L'ensemble $\mathcal{R}$ est exhaustif de manière jointe si $\forall R \in \mathcal{R} \mid \bigcup R = \divideontimes$ et il est disjoint deux à deux si $\forall R_i, R_j \in \mathcal{R}, i \neq j \mid R_i \cap R_j = \emptyset$. Les relations vérifiant la propriété JEPD sont dites relations basiques. Un ensemble de relations basiques

est noté $\mathcal{B}$. Par abus d'écriture on notera $\{R_i, R_j, R_k\}$ pour $R \cup S \cup T$. Si l'ensemble des relations formées par la génération de toutes les unions possibles des relations basiques est clos par composition et conversion alors cet ensemble supporte un algèbre de relation. Cet ensemble est noté $2^{\mathcal{B}}$.

2.6.2 L'algèbre de relations topologiques, RCC8

L'algèbre de relations RCC8(Region Connection Calculus)[81] permet de représenter des relations topologiques entre des objets dans un espace de dimensions entières. Les régions peuvent être composées de parties disjointes ou posséder des trous intérieurs. La relation de base est $C(x, y)$ elle lie deux régions x et y si leur fermeture topologique partage au moins un point. Les régions sont des classes d'équivalence d'ensemble de points dont la fermeture est identique. Dans RCC8, on définit huit relations de base. Il existe aussi d'autres algèbres de relations comme RCC5 qui utilise cinq relations, dans ce cas on ne peut pas distinguer si deux régions se touchent par l'extérieur, ou bien RCC23[28] qui permet de traiter des régions convexes.

Relation RCC8	Interprétation
$DC(x, y)$	x est déconnecté de y
$P(x, y)$	x est une partie de y
$PP(x, y)$	x est une partie propre de y
$EQ(x, y)$	x est identique à y
$O(x, y)$	x intersecte y
$DR(x, y)$	x n'intersecte pas y
$PO(x, y)$	x recouvre partiellement y
$EC(x, y)$	x est connecté par l'extérieur à y
$TPP(x, y)$	x est une partie propre tangentielle à y
$NTPP(x, y)$	x est une partie propre non tangentielle à y

TABLE 2.3 – Interprétation des relations définissables en terme de $C(x, y)$

RCC8 est composé des relations $\{DC, EC, PO, TPP, TPPi, NTPP, NTPPi\}$ les relations $TPPi$ et $NTPPi$ étant définies comme l'inverse de TPP et $NTPP$ respectivement. Chaque relation RCC8 possède ainsi un inverse. De plus RCC8 jouit de la propriété JEPD (Jointly Exhaustive and Pairwise Disjoint), à savoir : entre chaque région il existe une et une seule relation.

Relation	Définition
$DC(x,y)$	$\neg C(x,y)$
$P(x,y)$	$\forall z[C(z,x) \rightarrow C(z,y)]$
$PP(x,y)$	$P(x,y) \wedge \neg P(y,x)$
$EQ(x,y)$	$P(x,y) \wedge P(y,x)$
$O(x,y)$	$\exists z[P(z,x) \wedge P(z,y)]$
$DR(x,y)$	$\neg O(x,y)$
$PO(x,y)$	$O(x,y) \wedge \neg P(x,y) \wedge \neg P(y,x)$
$EC(x,y)$	$C(x,y) \wedge \neg O(x,y)$
$TPP(x,y)$	$PP(x,y) \wedge \exists z[EC(z,x) \wedge EC(z,y)]$
$NTPP(x,y)$	$PP(x,y) \wedge \neg\exists z[EC(z,x) \wedge EC(z,y)]$

TABLE 2.4 – Définition des relations définissables en terme de $C(x,y)$

On appelle RSAT le problème de satisfaction d'un réseau de contraintes RCC. Le problème CSPSAT consiste à décider de la consistance d'un ensemble de contraintes Θ. Le problème CSPMIN consiste à déterminer pour un ensemble de contraintes la relation la plus forte entre chaque paire de variables. Le problème CSPENT consiste à décider si une contrainte donnée peut être déduite d'un ensemble de contraintes. Ces trois problèmes sont réductibles les uns aux autres en temps polynomial.

RSAT est NP-complet mais il y a des sous-ensembles $\mathcal{S}$ de RCC-8 pour lesquels RSAT($\mathcal{S}$) est décidable en temps polynomial. On note $\widehat{\mathcal{S}}$ la fermeture d'un ensemble de relations $\mathcal{S}$ par composition, intersection et conversion. Il y a 256 unions possibles des relations de base. Pour l'ensemble des relations de base $\mathcal{B}$ et l'ensemble $\widehat{\mathcal{B}}$ (32 relations), RSAT($\mathcal{S}$) est décidable en temps polynomial. Les ensembles $\widehat{\mathcal{H}_8}$, $\mathcal{Q}_8$ (160 relations chacun) et $\mathcal{C}_8$ (158 relations) sont maximal par rapport à la décidabilité en temps polynomial. Chacun de ces ensemble contient $\mathcal{B}$. $\mathcal{NP}_8$ est l'ensemble des relations qui implique la NP-complétude quand elles sont combinées avec l'ensemble $\mathcal{B}$.

Il existe des formes de consistance moins fortes que l'on appelle consistance locale. Une variable est dite consistante par nœud si ses contraintes unaires sont satisfaites pour toutes les valeurs du domaine de la variable. Autrement on peut réduire le domaine de la variable. Une variable est consistante par arc avec une autre variable si chacune de ses valeurs admissibles est consistante avec au moins une valeur admissible de la seconde variable. Les variables x_i et x_j sont consistantes par chemins avec la variable x_k si

pour toute paire de valeurs (a, b) qui satisfont la contrainte entre x_i et x_j il existe une valeur c dans le domaine de x_k tel que (a, c) et (c, b) satisfont les contraintes entre x_i et x_k et entre x_k et x_j respectivement.

Un ensemble de contraintes RCC-8 sur n régions Θ est représenté par une matrice de relations $n \times n$ Pour obtenir la consistance par chemin, on effectue l'opération $M_{ij} \leftarrow M_{ij} \cap M_{ik} \circ M_{kj}$ jusqu'à l'obtention d'un point fixe $\overline{M}$. La consistance par chemins n'implique pas la consistance et réciproquement. Cependant la consistance par chemins permet de décider les problèmes RSAT($\widehat{\mathcal{H}_8}$), RSAT($\mathcal{Q}_8$) et RSAT($\mathcal{C}_8$)

2.6.3 L'algèbre des intervalles, IA

L'algèbre des intervalles[7] est principalement utilisé pour représenter des relations entre des intervalles temporels. Il peut toutefois être utilisé dans le cas spatial pour représenter les relations entre des éléments projetés ou disposés sur une droite, par exemple des éléments d'architecture disposés le long d'une façade. L'algèbre des intervalles est généré à partir d'un ensemble $\mathcal{B}_int$ de 13 relations de base entre des intervalles fermés.

Pour un réseau de contrainte de base la consistance par chemin est suffisante pour décider de la consistance.

2.6.4 L'algèbre des rectangles, RA

Il est aisé d'étendre l'algèbre des intervalles au cas bidimensionnel. Pour chaque région du plan, on définit un rectangle englobant r, puis on effectue les projections $I_r(x)$ et $I_r(y)$ sur les axes x et y respectivement. La relation de base entre deux rectangles a et b est $\alpha \otimes \beta$ ssi $(I_x(a), I_x(b)) \in \alpha$ et $(I_y(a), I_y(b)) \in \beta$ où α et β sont des relations IA de base. Il y a donc $13 \times 13 = 169$ relations de base. Tout comme pour l'algèbre des intervalles, la consistance par chemin implique la consistance.

2.6.5 L'algèbre de relations de directions, CDC

Une relation cardinale est une relation binaire entre un objet a et un objet de référence b. Dans le cas de CDC[38], on forme une grille de 3 par 3 autour de l'objet de référence b. Cette grille partitionne le plan en 9 zones b^{ij} $(1 \leq i, j \leq 3)$. La zone centrale est définie comme le rectangle englobant l'objet de référence. La relation δ_{ab} entre a et b est encodée dans une matrice booléenne $(d_{ij})_{1 \leq i,j \leq 3}$ de 3×3. La valeur de d_{ij} est 1 si l'objet intersecte

la zone b^{ij}. Une relation CDC peut être n'importe laquelle de ces matrices sauf la matrice nulle. Il y a donc $2^9 - 1 = 511$ relations de base. On note l'ensemble de ces relations $\mathcal{B}_{cdc}$.

2.6.6 Logiques descriptives et systèmes de contraintes

La logique descriptive $\mathcal{ALC}(\mathcal{C})$[62] permet d'associer un système de contraintes avec la logique descriptive $\mathcal{ALC}$. Le système de contrainte doit respecter la conditions d'être ω-admissible. Les intervalles de Allen et RCC8 respectent cette condition. La logique $\mathcal{ALC}(\mathcal{D})$[11] est utilisée pour associer le système de contraintes en tant que domaine concret.

Un domaine concret $\mathcal{D}$ est constitué d'un ensemble $dom(\mathcal{D})$ qui est le domaine de $\mathcal{D}$ et d'un ensemble de noms de prédicats $pred(\mathcal{D})$. Chaque nom de prédicat P est associé à une arité n et à un prédicat n-aire $P^{\mathcal{D}} \subseteq dom(\mathcal{D})^n$. A titre d'exemple, le domaine $\mathcal{D}$ peut être l'ensemble des polygones et $\{surface_>, surface_<\}$ l'ensemble des prédicats.

Pour que la logique $\mathcal{ALC}(\mathcal{D})$ soit décidable, il est nécessaire de mettre des restrictions sur les domaines concrets. Un domaine concret est dit *admissible* ssi

- l'ensemble des noms de prédicats est clos par la négation et contient un nom pour $dom(\mathcal{D})$;
- le problème de satisfiabilité pour les conjonctions finies de prédicats appartenant à $pred(\mathcal{D})$ est décidable.

L'ensemble des concepts de $\mathcal{ALC}(\mathcal{D})$ dans le cas de prédicats binaires est construit par la règle syntaxique suivante

$$C ::= A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C \mid \exists U_1, U_2.r \mid \forall U_1, U_2.r$$

où A est un concept atomique, R un rôle, r un prédicat binaire et U_1, U_2 des chaînes de rôles fonctionnels, tous les rôles de la chaîne étant abstrait, soit une fonction partielle de $\Delta_{\mathcal{I}}$ vers $\Delta_{\mathcal{I}}$, sauf le dernier rôle qui est concret, soit une fonction partielle de $\Delta_{\mathcal{I}}$ vers $dom(\mathcal{D})$. Dans le cas du système de contraintes RCC8, le prédicat binaire r appartient à l'ensemble $\{dc, ec, po, eq, tpp, ntpp, tppi, ntppi\}$ et le domaine concret à l'ensemble de variables $V_{M_{\mathcal{I}}}$.

Le système de contraintes utilisé dans le cas de la logique $\mathcal{ALC}(\mathcal{C})$ diffère un peu de celui décrit dans la section 2.6.1 puisqu'il contient un nombre de variables infini. Dans ce cas on définit un CSP de la manière suivante :

Soit Var un ensemble comptable infini de variables et Rel un ensemble de symboles de relations binaires. Une Rel-*contrainte* est une expression $(v\ r\ v')$ avec $v, v' \in \mathsf{Var}$ et $r \in \mathsf{Rel}$. Un Rel-*réseau* est un ensemble (fini ou infini) de Rel-contraintes. Pour un Rel-réseau N, l'ensemble des variables utilisées dans N est noté V_N. On dit que N est *complet* si, pour tout $v, v' \in V_N$, il existe exactement une contrainte $(v\ r\ v') \in N$. N est le *modèle* d'un réseau N' si N est complet et si il existe une application $\tau : V_{N'} \to V_N$ tel que $(v\ r\ v') \in N'$ implique que $(\tau(v)\ r\ \tau(v)') \in N$. Un CSP $\mathcal{C} = \langle \mathsf{REL}, \mathfrak{M} \rangle$ consiste en un un ensemble de symboles de relations binaires Rel et un ensemble $\mathfrak{M}$ de Rel-réseaux complets qui sont les modèles de $\mathcal{C}$.

On peut maintenant définir le système de contraintes $\mathsf{RCC8}_{\mathbb{R}^2} = \langle \mathsf{RCC8}, \mathfrak{M}_{\mathbb{R}^2} \rangle$ en définissant $\mathfrak{M}_{\mathbb{R}^2} := \{N_{\mathbb{R}^2}\}$. $N_{\mathbb{R}^2}$ est un Rel-réseau complet qui constitue le seul modèle du système de contraintes $\mathsf{RCC8}_{\mathbb{R}^2}$. On note $\mathcal{RS}_{\mathbb{R}^2}$ l'ensemble de toutes les régions non-vides et régulièrement fermées de $\mathbb{R}^2$. On définit une variable $v_s \in \mathsf{Var}$ pour tout $s \in \mathcal{RS}_{\mathbb{R}^2}$. L'ensemble des variables Var du modèle $N_{\mathbb{R}^2}$ est constitué de l'ensemble de ces variables.

2.7 Logiques métriques

2.7.1 Introduction

Les logiques métriques[57][58] permettent de traiter la notion de distance. Ces logiques s'appliquent sur un espace métrique $\langle W, d \rangle$ constitué d'un ensemble de points W et d'une fonction $d : W \times W \to \mathbb{R}^+$ qui satisfait :

$$d(x, y) = 0 \iff x = y \tag{2.7}$$

$$d(x, z) \leq d(x, y) + d(y, z) \tag{2.8}$$

$$d(x, y) = d(y, x) \tag{2.9}$$

Quelquefois il est utile de relâcher certains de ces axiomes. On définit ainsi les espaces de distances $\mathcal{D}$ satisfaisant seulement l'axiome (1), les espaces de distances symétriques satisfaisant $\mathcal{D}_{sym}$ (1) et (3) et les espaces de distances triangulaires $\mathcal{D}_{tr}$ satisfaisant (1) et (2).

2.7.2 La logique métrique du premier ordre $\mathcal{FM}[M]$

La logique $\mathcal{FM}[M]$[57] s'applique sur des distances appartenant à l'ensemble $M \subseteq \mathbb{R}^+$, l'ensemble M devant contenir 0. Le langage $\mathcal{FM}[M]$ fournit :

- un ensemble comptable infini de constantes $c_1, c_2, ...$
- un ensemble comptable infini de variables $x_1, x_2, ...$

- un ensemble comptable infini de symboles de prédicats unaires $P_1, P_2, ...$
- le symbole d'égalité $\doteq$
- deux ensembles (pouvant être infinis) de prédicats binaires de la forme : $\delta(\_,\_) < a$ et $\delta(\_,\_) = b$ $(a, b \in M)$
- les opérateurs booléens $\wedge$ (et), $\vee$ (ou), $\neg$ (négation), $\bot$ (falsum) et $\top$ (verum)
- les quantificateurs $\exists$ et $\forall$

Le problème de la satisfiabilité de la logique $\mathcal{FM}[M]$ est indécidable dans le cas général. Cependant en se restreignant à deux variables on obtient la logique $\mathcal{FM}^2[M]$ qui est décidable pour la classe des distances $\mathcal{D}$ et la classe des distances symétriques $\mathcal{D}_{sym}$ avec $M = \mathbb{Q}^+$ et $M = \mathbb{N}$.

2.7.3 Logique métrique du premier ordre $\mathcal{FM}^2[M]$

Le problème de satisfaisabilité d'une formule $\mathcal{FM}[\mathbb{Q}^+]$ ou $\mathcal{FM}[\mathbb{N}]$ est indécidable pour tout espace de distance contenant l'espace métrique $\mathcal{M}$. On définit $\mathcal{FM}^2[\mathbb{Q}^+]$ comme la logique $\mathcal{FM}[\mathbb{Q}^+]$ restreinte à deux variables. On sait que le fragment de la logique du premier ordre à deux variables est décidable[39]. On obtient les résultats suivants :

- $\mathcal{FM}^2[\mathbb{Q}^+]$ est décidable pour :
  - la classe $\mathcal{D}$ des espaces de distances,
  - la classe $\mathcal{D}_{sym}$ des espaces de distances satisfaisant l'axiome (3).
- $\mathcal{FM}^2[\mathbb{Q}^+]$ est indécidable pour :
  - la classe $\mathcal{M}$ des espaces métriques,
  - la classe $\mathcal{D}_{tr}$ des espaces de distances satisfaisant l'axiome (2).

2.7.4 Logique métrique modale $\mathcal{MS}[M]$

Comme alternative à $\mathcal{FM}[M]$, nous pouvons définir un langage purement propositionnel muni d'opérateurs de distance, d'une manière similaire à la logique modale. L'alphabet de $\mathcal{MS}[M]$ est constitué comme suit :

- une liste infinie de variables d'ensembles (ou de régions) $X_1, X_2, ...$
- une liste infinie de constantes de location $c_1, c_2, ...$
- un ensemble constant $\{c\}$ pour toute constante de location c
- les ensembles constants $\top$ et $\bot$
- les opérateurs booléens pour les ensembles de termes ($\sqcap$ et $\neg$) et pour les formules ($\wedge$ et $\neg$)
- les constructeurs d'ensemble de termes $\mathsf{E}^{<a}, \mathsf{E}^{>a}, \mathsf{E}^{=a}$ et $\mathsf{E}^{>a}_{<b}$ ainsi que leur duals $\mathsf{A}^{<a}, \mathsf{A}^{>a}, \mathsf{A}^{=a}$ et $\mathsf{A}^{>a}_{<b}$ avec $a, b \in M$ et $a < b$

2.7.5 Langage sans variables $\mathcal{MS}^{\#}[M]$

La logique $\mathcal{MS}^{\#}[M]$ se restreint aux opérateurs $E^{<a}$, $E^{>a}$, $A^{<a}$, $A^{>a}$. On peut montrer que $\mathcal{MS}^{\#}[\mathbb{Q}^{+}/\mathbb{N}]$ est décidable pour

- la classe $\mathcal{D}$ des espaces de distance,
- la classe $\mathcal{D}_{sym}$ des espaces de distance satisfaisant l'axiome (3),
- la classe $\mathcal{D}_{tr}$ des espaces de distance satisfaisant l'axiome (2),
- la classe $\mathcal{M}$ des espaces métriques.

Cependant $\mathcal{MS}^{\#}[M]$ est moins expressif que $\mathcal{MS}[M]$ et ne permet pas d'exprimer le concept d'anneau (cercles concentriques entre un rayon min et un rayon max).

2.7.6 Logiques descriptives et logiques métriques

Il est possible de combiner logiques descriptives et logiques métriques. Nous décrirons ici la logique $\mathcal{ALC}(\mathcal{MS})$[58].

Syntaxe

Au niveau de la syntaxe, les symboles de $\mathcal{ALC}(\mathcal{MS})$ sont ceux de $\mathcal{ALC}$ et de $\mathcal{MS}$ combinés. Pour $\mathcal{ALC}$, nous avons les noms de concepts $A_0, A_1, ..$, les noms de rôles $R_0, R_1, ...$, les noms d'individus $c_0, c_1, ...$ et les constructeurs de concepts $\sqcap, \neg, \top, \exists R_i$. Pour $\mathcal{MS}$ nous avons les variables d'ensemble $X_0, X_1, ...$, les nominaux $n_0, n_1, ...$ et les constructeurs de termes d'ensemble $\cap$, $\neg$, $E_{\leq\alpha}$ et $E_{\geq\alpha}$, pour tout $\alpha \in \mathbb{Q}_{+}$.
Le problème est ensuite de combiner les deux formalismes. Pour ce faire, on associe les éléments de $\mathcal{ALC}$ à ceux de $\mathcal{MS}$ par une relation binaire $\rightsquigarrow$ sur $\Delta \times D$, où Δ est le domaine de $\mathcal{ALC}$ et D celui de $\mathcal{MS}$. Si C est un $\mathcal{ALC}$-concept, son extension spatiale $\varsigma(C)$ est définie de la manière suivante :

$$\varsigma(C) = \{x \in D \mid \exists a \in C^{\mathcal{J}}.a \rightsquigarrow x\}$$

De la même manière, pour un terme d'ensemble t, on définit son extension conceptuelle $\varsigma^{-1}(t)$ par :

$$\varsigma^{-1}(t) = \{a \in \Delta \mid \exists x \in t^{\mathcal{D}}.a \rightsquigarrow x\}$$

Les concepts de $\mathcal{ALC}(\mathcal{MS})$ sont définis inductivement de la manière suivante :

- tous les noms de concepts A_i sont des concepts ;

– si C et D sont des concepts, R un nom de rôle et t un ensemble de termes alors

$$C \sqcap D,\ \neg C,\ \exists R.C,\ \varsigma^{-}(t)$$

sont des concepts.

Les termes d'ensemble de $\mathcal{ALC}(\mathcal{MS})$ sont définis inductivement de la manière suivante :

– Tous les nominaux et les variables d'ensemble sont des termes d'ensemble.
– Si s, t sont des termes d'ensemble, C un concept, c un nom d'individu, et $\alpha \in \mathbb{Q}_+$ un rationnel positif alors

$$t \cap s,\ \neg t,\ E_{\leq \alpha} t,\ E_{\geq \alpha} t,\ \varsigma(C),\ \varsigma(c)$$

sont des termes d'ensemble.

Les formules de $\mathcal{ALC}(\mathcal{MS})$ sont des combinaisons booléennes des formules atomiques de la forme

$$c : C,\ cRd,\ C \sqsubseteq D,\ t \sqsubseteq s$$

où C, D sont des concepts et t, s des termes d'ensemble.

Sémantique

Un modèle du langage $\mathcal{ALC}(\mathcal{MS})$[58] consiste en un modèle standard de $\mathcal{ALC}$, un modèle d'espace métrique pour $\mathcal{MS}$ et une relation entre les domaines, interprétant les fonctions d'extension spatiale et conceptuelle ς et ς^{-}.

Un modèle $\mathcal{ALC}(\mathcal{MS})$ est un triple de la forme $\mathfrak{M} = \langle \mathcal{J}, \mathcal{D}, \rightsquigarrow \rangle$ dans lequel

– $\mathcal{J} = \langle \Delta, R_0^{\mathcal{J}}, ..., A_0^{\mathcal{J}}, ..., c_0^{\mathcal{J}}, ... \rangle$ est un modèle $\mathcal{ALC}$, où Δ est un ensemble non-vide, le domaine des individus de $\mathfrak{M}$, $R_i^{\mathcal{J}}$ sont des relations binaires sur Δ interprétant les noms de rôles, $A_i^{\mathcal{J}}$ sont des sous-ensembles de Δ interprétant les noms de concept, et $c_i^{\mathcal{J}}$ sont des éléments de Δ interprétant les noms d'individus.
– $\mathcal{D} = \langle D, \delta, X_0^{\mathcal{D}}, ..., n_0^{\mathcal{D}}, ... \rangle$ est un modèle $\mathcal{MS}$, où $\langle D, \delta \rangle$ est un espace métrique, $X_i^{\mathcal{D}} \subseteq D$ sont les sous-ensembles de D interprétant les termes d'ensemble, et $n_i^{\mathcal{D}}$ sont les singletons de D interprétant les nominaux.
– $\rightsquigarrow$ est une relation binaire sur $\Delta \times D$.

L'extension $C^{\mathfrak{M}} \subseteq \Delta$ d'un concept C de $\mathcal{ALC}(\mathcal{MS})$ se fait de la manière suivante :

– $A_i^{\mathfrak{M}} = A_i^{\mathcal{J}}$, où A_i est un nom de concept,
– $(C_0 \sqcap C_1)^{\mathfrak{M}} = C_0^{\mathfrak{M}} \cap C_1^{\mathfrak{M}}$,
– $(\neg C_0)^{\mathfrak{M}} = \Delta \setminus C_0^{\mathfrak{M}}$,

– $a \in (\exists R_i.C_0)^{\mathfrak{M}}$ ssi il existe un $b \in C_0^{\mathfrak{M}}$ tel que $aR_i^{\mathcal{J}}b$,
– $a \in (\varsigma(t))^{\mathfrak{M}}$ ssi il existe un $x \in t^{\mathfrak{M}}$ tel que $a \rightsquigarrow x$.

L'extension $t^{\mathfrak{M}} \subseteq D$ d'un terme t de $\mathcal{ALC}(\mathcal{MS})$ se fait de la manière suivante :

– $X_i^{\mathfrak{M}} = X_i^{\mathcal{D}}$, où X_i est une variable d'ensemble,
– $n_i^{\mathfrak{M}} = n_i^{\mathcal{D}}$, où n_i est un nominal,
– $(t_0 \cap t_1)^{\mathfrak{M}} = t_0^{\mathfrak{M}} \cap t_1^{\mathfrak{M}}$,
– $(\neg t_0)^{\mathfrak{M}} = D \setminus t_0^{\mathfrak{M}}$,
– $(E_{\leq\alpha}t_0)^{\mathfrak{M}} = \{x \in D \mid \exists y \in D(\delta(x,y) \leq \alpha \wedge y \in t_0^{\mathfrak{M}})\}$,
– $(E_{\geq\alpha}t_0)^{\mathfrak{M}} = \{x \in D \mid \exists y \in D(\delta(x,y) \geq \alpha \wedge y \in t_0^{\mathfrak{M}})\}$,
– $x \in (\varsigma(C))^{\mathfrak{M}}$ ssi il existe un $a \in C^{\mathfrak{M}}$ tel que $a \rightsquigarrow x$,
– $x \in (\varsigma(c_i))^{\mathfrak{M}}$ ssi $c_i^{\mathcal{J}} \rightsquigarrow x$.

Décidabilité

Comme dans les logiques descriptives classiques la plupart des tâches de raisonnement peuvent être réduites au problème de satisfiabilité des formules $\mathcal{ALC}(\mathcal{MS})$. Pour $\mathcal{ALC}$ et pour $\mathcal{MS}$ le problème de satisfiabilité est décidable. Dans l'article[58], il est démontré que la logique $\mathcal{ALC}(\mathcal{MS})$ est décidable et que toute $\mathcal{ALC}(\mathcal{MS})$-formule ϕ possède un modèle fini dont la taille peut être calculée à partir de la longueur de ϕ.

2.8 $\mathcal{E}$-connection

Les $\mathcal{E}$-connection[59][12] permettent de lier différents formalismes logiques entre eux. Il est ainsi possible de lier des logiques descriptives, modales, spatiales ou temporelles entre elles. L'idée consiste à avoir des domaines d'interprétation disjoints pour chacun des formalismes et d'utiliser une application bijective pour établir une correspondance entre ces domaines. Le formalisme issu de la $\mathcal{E}$-connection entre deux formalismes consiste en l'union de ces deux formalismes plus un ensemble d'opérateur permettant de décrire la correspondance entre les domaines.

∘	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ
DC	*	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC	DC	DC
EC	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPP, TPPi, EQ	DC, EC, PO, TPP, NTPP	EC, PO, TPP, NTPP	PO, TPP, NTPP	DC, EC	DC	EC
PO	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPPi, NTPPi	*	PO, TPP, NTPP	PO, TPP, NTPP	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPPi, NTPPi	PO
TPP	DC	DC, EC	DC, EC, PO, TPP, NTPP	TPP, NTPP	NTPP	DC, EC, PO, TPP, TPPi, EQ	DC, EC, PO, TPPi, NTPPi	TPP
NTPP	DC	DC	DC, EC, PO, TPP, NTPP	NTPP	NTPP	DC, EC, PO, TPP, NTPP	*	NTPP
TPPi	DC, EC, PO, TPPi, NTPPi	EC, PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPP, TPPi, EQ	PO, TPP, NTPP	TPPi, NTPPi	NTPPi	TPPi
NTPPi	DC, EC, PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPP, NTPP, TPPi, NTPPi, EQ	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ

TABLE 2.5 – Tableau de composition des relations RCC8

Relation	Symbole	Converse	Signification
precedes	p	pi	$x^+ < y^-$
meets	m	mi	$x^+ = y^-$
overlaps	o	oi	$x^- < y^- < x^+ < y^+$
starts	s	si	$x^- = y^- < x^+ < y^+$
during	d	di	$y^- < x^- < x^+ < y^+$
finishes	f	fi	$y^- < x^- < x^+ = y^+$
equals	eq	eq	$x^- = y^- < x^+ = y^+$

TABLE 2.6 – L'ensemble des relations de bases $\mathcal{B}_{int}$, où $x = [x^-, x^+]$, $y = [y^-, y^+]$ sont des intervalles

Chapitre 3

Logiques et architecture

Dans ce chapitre, les différentes logiques présentées précédemment vont être illustrées à l'aide d'exemples issus de l'architecture. Il ne s'agit pas encore de présenter le modèle des éléments de conception architecturale, mais simplement d'examiner en quoi ces différentes logiques peuvent être utiles dans le cadre de la représentation architecturale.

Modéliser les éléments de conception architecturale nécessite de pouvoir définir ces éléments de manière d'abord seulement partielle. Lors de la conception, les éléments seront déterminés de plus en plus complètement par une succession d'opérations de raffinement et de retour en arrière. La logique descriptive et son hypothèse du monde ouvert est particulièrement adaptée à cela.

La modélisation se fera donc, autant que possible, en logique descriptive. Ceci afin d'utiliser les moteurs de raisonnement standard disponibles. Les logiques spatiales métriques et topologiques seront aussi utilisées. Elles seront, dans la mesure du possible, inter-connectées avec la logique descriptive.

3.1 Outils d'aide à la conception architecturale

L'architecture a d'abord été liée au calcul par le biais de la géométrie, qui permettait de déterminer les tailles, les proportions, les surfaces et les volumes. La règle et le compas étaient les premiers outils d'aide à la conception architecturale, permettant de tendre vers l'idéal vitruvien[78] de l'inscription des proportions humaines dans la proportion des édifices (Fig.3.1). Par la suite, le développement de la représentation en perspective, systématisée par Leon Battista Alberti[6] en 1453, permit pour la première fois de représenter visuellement des bâtiments en volume. Ce n'est qu'à partir de 1757, grâce

au mathématicien bâlois Leonhard Euler, que les fondements mathématiques permettant le développement de la science des matériaux et de la statique des bâtiments, ouvrirent l'ère de l'utilisation du calcul numérique en architecture.

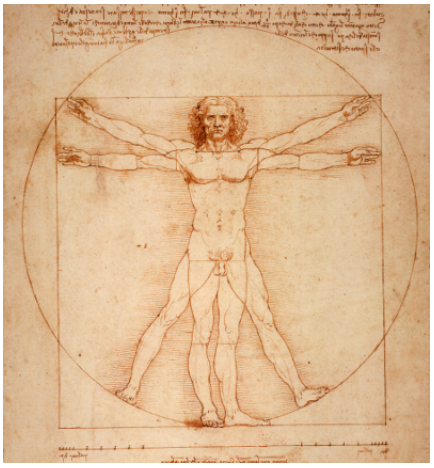


FIGURE 3.1 – L'homme de Vitruve selon Leonard de Vinci, 1490

L'histoire des outils de conception assistée par ordinateur (CAO) peut être vue comme la succession de trois générations[53]. La première génération émergea dans le courant des années 1970, suivant les traces des pionniers comme Ivan Sutherland et son logiciel Sketchpad[94] (1963). Deux voies différentes furent développées. D'un coté, la modélisation géométrique, soutenue par les géants de l'industrie automobile et aérospatiale, qui permettait le dessin de courbes complexes et la construction géométrique par des opérations booléennes. D'un autre coté, des systèmes soutenus par le secteur académique, plus spécifiques à l'architecture, permettant la description de bâtiments et la planification de l'espace. On peut citer notamment le système BDS (Building Description System) contenant une base de données spécifique à l'architecture et un modeleur géométrique. Son successeur, GLIDE[27] (Graphical Language for Interactive Design), était quant à lui basé sur un langage de programmation interprété supportant des opérations géométriques et des descriptions paramétriques de bâtiments.

La deuxième génération d'outils CAO émergea dans les années 1980 avec l'arrivée des ordinateurs personnels. Contrairement à la première génération qui resta confinée dans les centres de recherche, la seconde génération d'outils CAO fit son entrée dans la plupart des bureaux d'architectes. Cependant de

manière paradoxale, cette seconde génération fut circonscrite à un outil de dessin et laissa de côté les avancées en termes d'assistance à la conception.

La troisième génération d'outils CAO réintroduit la notion d'attributs non-géométriques. Le système KAAD[30] (Knowledge-Assisted Architectural Design), par exemple, est basé sur une modélisation par *frames* et implémenté en LISP. Il permet l'introduction de règles et leur vérification automatique. Le système ICAAD, quant à lui est basé sur un système multi-agents, chaque agent étant responsable d'une partie de la conception et dialoguant avec les autres agents par des propositions et des évaluations. Cette troisième génération, comme la première, a la vocation d'aider à la conception et non de seulement représenter les bâtiments. La différence provient des avancées réalisées en deux décennies dans les domaines de la programmation orientée objets (POO), de l'intelligence artificielle (IA) et des systèmes de gestion de bases de données (SGBD).

Selon Kalay[53], le processus de conception architecturale a pour objectif la création d'une forme contrainte par sa fonction et par son contexte. Ce processus se divise en trois étapes : l'analyse, la synthèse et l'évaluation. Chacune de ces étapes doit pouvoir être communiquée aux différents participants de ce processus. L'analyse a pour but d'identifier les éléments du problème, notamment les objectifs à atteindre, les contraintes à respecter ainsi que les effets de bord que la solution pourrait générer. La synthèse est la phase où interviennent la créativité et le hasard. Il n'existe en général pas de méthodes permettant d'atteindre de manière certaine les objectifs. Comme il est impossible de séparer l'activité conduisant à la compréhension du problème et l'activité conduisant à trouver une solution, la synthèse implique souvent la redéfinition des contraintes et des objectifs. L'évaluation, quant à elle, permet de vérifier qu'une solution respecte les contraintes, atteigne les objectifs et soit cohérente. Quand les objectifs ou les contraintes ne sont pas respectés, l'évaluation doit pouvoir déterminer la distance entre la solution et les objectifs à atteindre, et même proposer des modifications. Certains objectifs ne sont toutefois pas quantifiables et seul un humain peut les évaluer (par exemple une salle de séjour qui soit agréable à vivre). Il arrive aussi que des objectifs soient contradictoires et que des compromis doivent être trouvés. Quant à la communication, elle est intimement liée à la représentation du problème et de ses solutions partielles. Les sciences informatiques définissent six propriétés désirables pour les représentations :

- *La flexibilité* permettant de changer de niveau d'abstraction sans avoir à reconstruire la représentation depuis zéro.
- *La covariance* permettant de lier des informations de manière à ce que,

quand une représentation est modifiée, les autres représentations le soient aussi.
- *La gestion d'information* permettant d'organiser et d'accéder à des bases d'informations complexes.
- *La visualisation* permettant de produire une représentation visuelle à partir d'une représentation abstraite.
- *L'intelligence* permettant d'inclure des règles de conception, des contraintes et des objectifs dans la représentation elle-même, faisant de cette représentation un objet actif plutôt que passif.
- *La connectivité* permettant de partager rapidement de l'information entre tous les participants au processus de conception.

Dans son ouvrage de 1969, intitulé *The Sciences of the Artificial*[87], Herbert Simon définit, quant à lui, la conception comme *un processus de prise de décisions contraint par la physique, la logique et la cognition.* La conception était alors vue alors comme une activité tendant à résoudre des problèmes qui avaient la particularité d'être mal définis. Dans son article *Artificial intelligence in designing*[36], John S. Gero retrace l'évolution des systèmes d'aide à la conception de ces vingt dernières années. Dans les années 1980, le paradigme de l'intelligence artificielle fournissait un cadre pour explorer des espaces de solutions dans le domaine de la conception assistée. Les technologies étaient basées sur les systèmes experts, les systèmes à base de contraintes, la programmation logique, le raisonnement basé sur des cas. Plus récemment, les algorithmes génétiques, les réseaux de neurones et la logique floue ont été appliqués. Ces systèmes d'aide à la conception ont hérité des paradigmes de systèmes plus anciens basés sur la recherche opérationelle et l'optimisation mathématique. La grande avancée de l'intelligence artificielle a été d'étendre l'espace de recherche des représentations numériques aux représentations symboliques. Cette avancée a permis notamment de travailler sur des représentations par configurations, comme par exemple les grammaires de formes. Stiny[91], quant à lui, utilise une approche visuelle des formes pour l'appliquer aux problèmes de conception. Terzidis[97], de son coté, se rapproprie le mode de pensée algorithmique pour générer une nouvelle forme d'architecture, qu'il appelle algotecture.

Les études cognitives portant sur les concepteurs et les conceptrices montrent que la reformulation des objectifs et des contraintes est une activité importante du processus de conception. De plus, les résultats intermédiaires ont une forte influence sur la suite de ce processus. Ces deux aspects ne sont pas pris en compte dans les méthodes par exploration de l'espace des solu-

tions. Ces nouvelles considérations soulèvent une nouvelle série de questions :

- Comment prendre en compte un espace de solutions qui change avec le temps ?
- Comment prendre en compte le fait que ce qui a été récemment modifié façonne le reste de la conception ?
- Comment prendre en compte la connaissance acquise durant le processus de conception ?
- Comment prendre en compte des objectifs qui changent durant le processus ?

Les récentes avancées dans le domaine des logiques descriptives et des logiques spatiales, notamment en terme de décidabilité, permettent de définir plus finement, et ce de manière formelle, la forme, la fonction et les relations entre la forme et la fonction[14]. Le principe architectural sous-jacent est le fonctionnalisme basé sur l'idée que *la forme dérive de la fonction*[93] et que *l'ornement est un crime*[61]. Le domaine d'application de l'article cité[14] étant le développement de bâtiments « intelligents », à savoir des bâtiments équipés de senseurs (caméras, détecteurs de mouvements, de luminosité, de fumée...) et d'actuateurs (alarmes, portes, stores automatiques...). Le modèle utilisé est basé sur l'articulation d'une ontologie modulaire[92] et d'une représentation de l'espace par logiques qualitatives[19]. Les différents formalismes logiques sont connectés par le biais des $\mathcal{E}$-connections[59].

Les travaux de Pranovich et al.[79] se concentrent sur les phases de la conception en amont de la représentation de l'objet final. Pour représenter la structure de la conception dès les premiers pas du processus, les auteurs utilisent des objets graphiques comme des trames et des axes de symétries. Les relations entre ces objets permettent de modéliser la conception avec de hauts niveaux d'abstraction. La représentation sous-jacente est basée sur une modélisation géométrique par contraintes et n'inclut pas de relations avec une base de connaissance. Les systèmes d'aide à la conception architecturale par base de connaissance ont été développés notamment à des fins d'enseignement[35]. Les systèmes à base de connaissance permettent l'acquisition, la représentation et la manipulation de connaissances explicites représentées sous forme symbolique. Ces systèmes permettent d'inférer des nouvelles connaissances à partir des connaissances actuelles et de l'état actuel du design. Ils permettent aussi d'expliquer comment ces nouvelles connaissances ont été inférées.

3.2 Logiques descriptives et architecture

La logique descriptive permet de décrire les différentes parties constituant un édifice et les relations que ces parties entretiennent entre elles. Les concepts de logique descriptive permettent de représenter les classes de constituants d'un bâtiment. Commençons par définir les concepts atomiques suivants :

$$Ouverture, Fermeture, Porte, Mur, Fenetre, Piece, Etage, Escalier$$

Ces concepts sont reliés entre eux par des relations de subsomption. Chaque individu appartenant au concept Mur appartient aussi au concept $Fermeture$ et chaque individu appartenant au concept $Porte$ appartient aussi au concept $Ouverture$. Dans le formalisme de la logique descriptive nous écrivons :

$$Mur \sqsubseteq Fermeture$$

$$Porte \sqsubseteq Ouverture$$

Pour l'instant, un individu appartenant au concept $Ouverture$ peut appartenir à la fois au concept $Porte$ et au concept $Fenetre$. Pour définir les concepts $Porte$ et $Ouverture$ comme disjoints il faut rajouter l'axiome :

$$Porte \sqcap Fenetre \equiv \bot$$

Sinon il est aussi possible de définir une $PorteFenetre$

$$PorteFenetre \equiv Porte \sqcap Fenetre$$

comme étant l'ensemble des individus appartenant aux deux concepts. Pour exprimer le fait que $Piece$ est une partie d'un $Etage$, il faut définir un nouveau rôle $estPartieDe$.

$$Piece \equiv \exists estPartieDe.Etage$$

Il est possible de définir le rôle inverse :

$$aPourPartie \equiv estPartieDe^{-1}$$

On peut ainsi exprimer :

$$Etage \equiv \exists aPourPartie.Piece$$

Ainsi un $Etage$ doit forcément posséder une $Piece$ et une $Piece$ doit forcément faire partie d'un $Etage$. Un $Batiment$ peut être défini par :

$$Batiment \equiv \exists aPourPartie.Etage$$

L'expansion de ce concept donne :

$$Batiment \equiv \exists aPourPartie.(\exists aPourPartie.Piece)$$

Le rôle $aPourPartie$, plus couramment appelé $hasPart$ définit ainsi, comme pour la relation de subsomption (aussi appelée IsA), un treillis de type meronymique. Pour définir une $Piece$, nous pouvons dire qu'elle est composée uniquement d'$Ouverture$ et de Mur :

$$Piece \equiv \exists hasPart.(Ouverture \sqcup Mur)$$

Si nous voulons associer à la pièce une surface en mètres carré, il faut utiliser un domaine concret sur les nombres rationnels à précision finie.

$$Piece \equiv \exists hasPart.(Ouverture \sqcup Mur) \sqcap \exists hasSurface$$

En OWL-DL, cela est dénommé *DatatypeProperty*. A partir de OWL 1.1, il est possible d'utiliser des XSD facets[16] pour, par exemple, contraindre la surface de la $Piece$. En utilisant la syntaxe de Protégé, appelée aussi syntaxe de Manchester, cela donne :

$$\texttt{Piece and hasSurface some int}[>= \texttt{50}, <= \texttt{120}]$$

Pour associer à $Piece$ une description sous forme textuelle, il faut lui ajouter une *DatatypeProperty* de type string. Toujours avec les XSD facets, il est possible de définir un concept avec une expression régulière sur la chaîne de caractère. En syntaxe Manchester :

$$\texttt{Piece and hasDescription some string}[*\texttt{red}*]$$

définit l'ensemble des individus de type $Piece$ dont la description contient la suite de caractère `red`. En OWL, les ObjectProperties permettent de lier des objets à $Piece$, en terminologie DL on parle de rôle.

$$Piece \sqcap \exists hasPart.Ouverture\forall \sqcap hasMur.MurEnBrique$$

Ces rôles sont organisés dans une hiérarchie de rôles, aussi appelé treillis de subsomption :

$$hasMur \sqsubseteq hasPart$$

La propriété universelle U relie tous les individus avec tous les individus. En OWL elle se note owl :topObjectProperty. La propriété vide $\emptyset$ qui ne relie aucun individu se note en OWL owl :bottomObjectProperty. Ainsi :

$$\emptyset \sqsubseteq hasMur \sqsubseteq hasPart \sqsubseteq hasPart \sqsubseteq U$$

Les chaînes de propriétés doivent être manipulées avec précaution car certaines combinaisons peuvent mener à l'indécidabilité. Les chaînes de propriétés doivent être régulières.[74]

Un mur peut « faire un angle droit »avec un autre mur. Il peut aussi être lié à des valeurs numériques comme des propriétés physiques (dimensions, coefficients thermiques, ...) ou des coefficients d'élasticité utilisés par le logiciel de conception. Il est aussi possible de les lier à un document virtuel ou à une adresse internet contenant les caractéristiques techniques d'un élément, et même d'intégrer automatiquement ces caractéristiques techniques dans la base de connaissance, si le fournisseur a publié un modèle de données standard.

La difficulté de ce travail ne consiste pas à représenter un bâtiment fini, mais de représenter un bâtiment en cours de conception, de manière à ce que les moteurs de raisonnement et les humains puissent collaborer à ce processus. La représentation peut être sur-spécifiée de sorte que des contraintes soient contradictoires. Elle peut être exactement spécifiée de sorte à ce qu'il n'y ait rien à enlever ni à ajouter pour obtenir un bâtiment constructible ; ou bien elle peut être sous-spécifiée.

3.2.1 Mondes ouverts et Mondes fermés

L'hypothèse du monde fermé (Close World Assumption, CWA) implique que si un individu ou une propriété n'est pas affirmée explicitement alors cette propriété est fausse. *A contrario*, l'hypothèse du monde ouvert (Open World Assumption, OWA) implique que si un individu ou une propriété n'est pas affirmée explicitement alors on ne peut rien affirmer sur l'existence ou non de cet individu ou de cette propriété. En logique descriptive, l'hypothèse par défaut est celle du monde ouvert.

3.2.2 Une technique de fermeture

Dans le cadre de cette thèse, a été développé un logiciel en JAVA, permettant de simuler l'hypothèse du monde fermé dans une base de connaissance en logique descriptive. Pour chaque individu et pour chacune de ses relations est ajouté un axiome de restriction exacte de cardinalité. Par exemple, si dans la base de connaissance, la pièce p_1 contient deux fenêtres alors un axiome disant que p_1 contient exactement deux fenêtre est ajouté. Sans cela, avec l'hypothèse du monde ouvert, la pièce pourrait contenir d'autres fenêtres non présentes dans la base de connaissance. Il devient ainsi possible de classifier la pièce dans le concept pièce de moins de 5 fenêtres et dans le concepts pièces dont toutes les ouvertures sont des fenêtres. Ce qui n'était pas possible avant.

En utilisant la notation de la logique descriptive, on peut dire que si nous avons la ABox :

$$Piece(p), Fenetre(f_1), Fenetre(f_2), hasPart(p_1, f_1), hasPart(p_1, f_2)$$

alors l'axiome suivant est automatiquement ajouté :

$$=_2 hasPart.Fenetre(p)$$

L'implémentation a été réalisée en utilisant la librairie de Protégé 3. L'algorithme principal est le suivant :

```
//For each known individual
for(OWLIndividual ind : ont.getReferencedIndividuals()) {
   Map<OWLObjectPropertyExpression,Set<OWLIndividual>> mapProperty =
      ind.getObjectPropertyValues(ont);
   //For each of its properties
   for(OWLObjectPropertyExpression prop : mapProperty.keySet()){
      //Count the number of fillers
      Set<OWLIndividual> setInd = mapProperty.get(prop);
      int nbInd = setInd.size();
      //assert an exact restriction with this number
      OWLObjectCardinalityRestriction restriction =
         dataFactory.getOWLObjectExactCardinalityRestriction(prop, nbInd);
      dataFactory.getOWLClassAssertionAxiom(ind, restriction);
      OWLClassAssertionAxiom classAxiom =
         dataFactory.getOWLClassAssertionAxiom(ind, restriction);
      AddAxiom addAxiomClass = new AddAxiom(ont, classAxiom);
      //annotate the added axiom
      OWLAnnotation annotation = dataFactory.getCommentAnnotation(strAnnotation);
      OWLAxiomAnnotationAxiom annotateAxiom =
         dataFactory.getOWLAxiomAnnotationAxiom(classAxiom, annotation);
      AddAxiom addAxiomAnnotate = new AddAxiom(ont,annotateAxiom);
      try{
         manager.applyChange(addAxiomClass);
         manager.applyChange(addAxiomAnnotate);
         manager.saveOntology(ont, URI.create(strOntURI));
      }catch(Exception e){
         e.printStackTrace();
      }
   }
}
```

Il est aussi possible d'utiliser l'hypothèse du monde fermé en définissant le concept `owl:Thing` comme équivalent à une énumération de l'ensemble des individus présents dans la ABox.

3.3 Logiques métriques et architecture

La logique métrique $\mathcal{FM}^2[M]$ définie dans la section 2.7.3 permet de spécifier des contraintes de distance entre des éléments d'architecture. Les contraintes suivantes peuvent, par exemple, être définies :

- Les portes en bois doivent être éloignées de 2 mètre au moins de la (ou les) cheminée(s).
  $\forall x \forall y(PorteEnBois(x) \wedge Cheminee(y) \rightarrow \delta(x, y) > 2)$

- Il doit y avoir des sanitaires à moins de 8 mètres de chaque chambre.
  $\forall x \exists y(Chambre(x) \wedge Sanitaire(y) \rightarrow \delta(x, y) < 8)$

- Toutes les ouvertures doivent être distantes d'au moins 1 mètre.
  $\forall x \forall y(Ouverture(x) \wedge Ouverture(y) \wedge x \neq y \rightarrow \delta(x, y) > 1)$

Les distances entre les objets ne nécessitent pas de respecter l'inégalité triangulaire. Ainsi dans un espace à trois dimensions, la distance entre deux pièces peut être définie comme la longueur du chemin nécessaire pour rejoindre une pièce depuis une autre et non pas comme la distance directe dans l'espace 3D.

3.4 Logiques topologiques et architecture

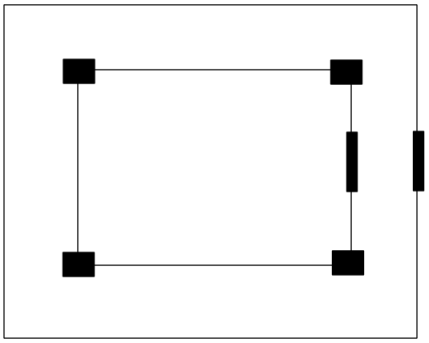


FIGURE 3.2 – Représentation de forme par RCC8

Le formalisme RCC8, défini dans la section 2.6.2, peut être utilisé pour représenter la topologie des bâtiments.
Pour représenter des ouvertures qui lient deux régions, nous procédons de la manière suivante : si la région A et la région B sont séparées par un mur

FIGURE 3.3 – Porte d'enceinte de La Tour Nord de la Couvertoirade (Aveyron)

alors on utilise la relation EQ. Par contre si les deux régions sont séparées par une ouverture, comme par exemple une porte, alors on utilise la relation PO. On peut imaginer que le pas de la porte est recouvert à la fois par la région A et par la région B, ainsi les régions A et B sont liées par la relation PO à la porte.

La figure 3.2 est une représentation du plan d'un bâtiment constitué d'une enceinte, d'une cour intérieure, d'une zone extérieure, de deux portes et de quatre piliers. Les relations RCC8 suivantes permettent de définir de manière complète l'ensemble des relations entre la cour, l'enceinte, la zone extérieure et les portes :

- $PO(cour, porte1)$
- $PO(enceinte, porte1)$
- $PO(enceinte, porte2)$
- $PO(zone\_extérieure, porte2)$
- $EC(cour, enceinte)$
- $EC(enceinte, zone\_extérieure)$
- $DC(porte1, porte2)$
- $DC(cour, zone\_extérieure)$
- $DC(porte1, zone\_extérieure)$

– $DC(cour, porte2)$

Toutes les relations sont spécifiées puisque avec 5 objets il faut $\frac{5\times4}{2} = 10$ relations.

Le logiciel *SparQ*[102][24] permet d'appliquer l'algorithme de consistance par chemin. La configuration de scène est exprimée en notation LISP de la manière suivante :

```
((cour PO porte1) (enceinte PO porte1)
 (enceinte PO porte2) (zone_extérieur PO porte2)
 (cour EC enceinte) (enceinte EC zone_extérieur)
 (porte1 DC porte2) (cour DC zone_extérieur)
 (porte1 DC zone_extérieur) (cour DC porte2))
```

En utilisant la requête :

```
constraint-reasoning rcc8 scenario-consistency all
```

On peut vérifier qu'il n'existe qu'un seul scénario respectant l'ensemble des contraintes définies ci-dessus.
Suivant les interprétations, l'enceinte de la figure 3.3, peut être vue comme étant traversée par une seule porte, ou bien par deux portes. En retirant l'axiome qui affirme que les deux portes sont déconnectées $DC(porte1, porte2)$ et en ajoutant un axiome connectant chacune des portes sur leur zone opposée respective $PO(zone\_extérieur, porte1), PO(cour, porte2)$, on obtient un nouvel ensemble de neuf relations qui s'exprime sous la forme :

```
((cour PO porte1) (enceinte PO porte1)
 (enceinte PO porte2) (zone_extérieur PO porte2)
 (cour EC enceinte) (enceinte EC zone_extérieur)
 (cour DC zone_extérieur) (zone_extérieur PO porte1)
 (cour PO porte2))
```

En appliquant le raisonneur *SparQ*, on obtient 8 scénarios possibles, dont chacun définit un type de relations différent entre la porte 1 et la porte 2.

3.4.1 Application de Pellet Spatial

Pellet Spatial[88] est un moteur de raisonnement spatial permettant d'effectuer des requêtes SPARQL sur des données combinant des relations RCC8 et RDF/OWL.

Dans notre cas, on définit des object properties comme des sous-propriétés des relations RCC8, par exemple on définit la relation *borders* comme sous-propriété de la relation RCC8 *externallyConnectedTo*.

En utilisant cette définition, on peut rechercher, dans une base de connaissance, tous les objets ?x qui sont extérieurement connectés à la maison bleue numéro 4 et tous les objets ?y qui recouvrent ces objets ?x.

```
PREFIX spatial: <http://clarkparsia.com/pellet/spatial#>
PREFIX ex: <http://example.org/test#>

SELECT ?x ?y
WHERE {
        ?x ex:borders ex:blueHouse4 .
        ?x spatial:partiallyOverlaps ?y
}
```

On peut classer par superficie toutes les entités de la base de connaissance qui sont contenues dans le territoire des états-unis.

```
PREFIX spatial: <http://clarkparsia.com/pellet/spatial#>
PREFIX us: <http://example.org/usastates#>

SELECT ?name ?area
WHERE {
        us:000 us:completelySpatiallyContains ?x .
        ?x us:hasArea ?area .
        ?x us:hasName ?name .
}
ORDER BY DESC(?area)
```

On peut lister toutes les personnes travaillant dans un local avec une porte donnant sur le couloir de l'aile Jura au 3ème étage et tous les types de ces locaux présents dans la base de connaissance.

```
PREFIX spatial: <http://clarkparsia.com/pellet/spatial#>
PREFIX batelle: <http://cui.unige.ch/isi/batelleArchi#>

SELECT ?personne ?type
WHERE {
        ?personne rdf:type batelle:Membre .
        ?personne batelle:aCommeBureau ?local .
        ?local spatial:PO batelle:couloirJura3 .
        ?local rdf:type ?type .
}
```

On peut lister toutes les paires de pièces liées par un couloir non chauffé.

```
PREFIX spatial: <http://clarkparsia.com/pellet/spatial#>
PREFIX batimentABC: <http://cui.unige.ch/isi/batimentABC#>
PREFIX archiKB: <http://cui.unige.ch/isi/archiKB#>

SELECT ?piece1 ?piece2
WHERE {
_:couloir rdf:type archiKB:Couloir .
?piece1 spatial:PO _:couloir .
?piece2 spatial:PO _:couloir .
_couloir rdf:type archiKB:NonChauffé
}
```

3.5 Représentation géométrique d'une forme

3.5.1 Encodage de formes

Une forme est, ici, représentée par sa description géométrique. Pour effectuer des raisonnements sur cette forme, il est nécessaire de mettre en adéquation la représentation de cette forme avec l'application envisagée. Il est parfois nécessaire de réduire la complexité de cette description. Cette réduction peut se faire, dans certain cas, sans perte d'information. Dans d'autres cas, la complexité de la représentation est réduite avec perte d'information. Des formes géométriquement proches deviennent alors identiques. Dans ces cas, la représentation induit des classes d'équivalence de formes. Il est souvent utile d'associer à ces classes d'équivalence un représentant unique qui est exprimé de manière canonique.

Les relations entre les représentants correspondent à des relations entre les classes de formes. Les relations entre des formes qui sont significatives pour une application donnée doivent correspondre à des relations facilement exprimables par le biais de la description géométrique choisie. Dans le cas de l'architecture, le choix des caractéristiques descriptives d'une forme est intimement lié au style architectural ou à la manière de concevoir de l'architecte.

L'encodage d'une figure par un ensemble de caractéristiques organisées sous la forme d'un vecteur ou d'un graphe de relations nécessite de déterminer s'il existe une application bijective entre les formes et leurs représentations. S'il existe plusieurs formes associées à une même représentation, alors il faut faire appel aux classes d'équivalence et à leurs représentants pour obtenir une

application bijective entre les classes d'équivalence et les représentations.

3.5.2 Graphes et formes

Graphes sur des formes

Une forme peut être représentée par un graphe en faisant correspondre des éléments de cette forme avec les nœuds du graphe de la manière suivante :

- faire correspondre les angles de la forme avec des nœuds du graphe (A-node),
- faire correspondre les segments de la forme avec des nœuds du graphe (L-node),
- faire correspondre les surfaces de la forme avec des nœuds du graphe (S-node),

Il est ensuite possible d'ajouter des étiquettes sur les arrêtes (type de connexité,...) ou sur les nœuds (nombre de surfaces couvrantes, valeurs d'angle, convexité ...)

Le codage nécessite un ensemble de règles syntaxiques qui définit l'ensemble des suites de symboles ou des configurations de graphes qui représentent effectivement des formes. Certains types de codage demandent que le graphe possède un chemin hamiltonien (chemin passant par tous les sommets une seule fois) ou un chemin euclidien (chemin passant par toutes les arrêtes une seule fois).

Graphes et règles

Les règles SWRL[48] permettent d'étendre la sémantique des logiques descriptives. Alors que la DL ne permet de détecter que des chemins dans une ABox, l' ajout de règles permet la détection de patterns contentant des combinaisons de chemins et de cycles. Et ce en vérifiant pour chaque instance de concept et de rôle s'ils appartiennent au concept ou rôle désiré.

En utilisant des règles, il est possible d'étiqueter un individu en fonction de son appartenance et de son rôle dans un sous-graphe.

Passage d'un graphe à une géométrie

Pour faciliter le passage d'un graphe à une géométrie, il est possible d'ajouter dans le graphe des contraintes sur les formes géométriques, comme par exemple les aires admissibles minimales et maximales pour chaque pièce, certaines caractéristiques de formes et les types de connexité entre pièces. Le

problème est que la partie purement logique du système ne peut pas vérifier qu'il existe une solution géométrique. De plus, s'il existe une solution, il y en a certainement plusieurs, voire une infinité. Il faut donc qu'il y ait une interaction bidirectionnelle entre la partie du système purement logique et la partie du système traitant de la géométrie. Afin d'avoir des temps de calculs raisonnable, il est peut-être nécessaire de faire en sorte que l'ajout ou le retrait d'une contrainte n'implique pas le recalcul complet de la géométrie mais un calcul incrémental à partir de la solution précédente.

De la classification à la décomposition des formes

En utilisant les formalismes logiques, la description des formes géométriques, la description des fonctions associées à ces formes ainsi que des relations existant entre ces formes et ces fonctions peuvent être stockées dans une base de connaissance. Les formes géométriques peuvent être décrites par leurs propriétés comme par exemple le périmètre, l'ellipsoïde d'inertie ou le rectangle englobant. Les formes géométriques peuvent aussi être décomposées en un ensemble de formes plus simples, en relation les unes avec les autres. Ces relations forment un graphe qui à son tour peut être décrit en utilisant les propriétés mathématiques des graphes. Comme il existe de nombreuses manières différentes d'effectuer l'opération de décomposition d'une forme, il est nécessaire d'avoir une connaissance *a priori* du type de décomposition pertinent pour un contexte donné. Le graphe de décomposition peut ainsi être classifié dans une ontologie et être sujet à des manipulations par des règles logiques. Le graphe de décomposition de forme étant stocké dans la base de connaissance, chacun de ses sommets peut être annoté par des concepts et des instances terminologiques.

Au niveau du formalisme, les relations spatiales entre les formes sont encodées soit par le biais de logiques spatiales spécifiques, comme les logiques métriques ou les logiques topologiques, ce qui permet d'utiliser des algorithmes de raisonnement dédié au spatial ; soit par une logique de type terminologique qui utilise uniquement des axiomes relatifs à l'espace, propres à une application donnée.

3.5.3 Représentation par relations géométriques

Représentation par prédicats géométriques

Il s'agit de représenter une configuration géométrique dans un espace de dimension n par un ensemble d'assertions unaires et binaires. Cet ensemble d'assertions forme une ABox et les outils de la logique descriptive peuvent lui

être appliqué. Cependant pour vérifier que cet ensemble d'assertions est consistant dans l'espace, il est nécessaire d'utiliser des algorithmes géométriques. En utilisant des techniques d'optimisation sous contraintes, il est possible d'associer des valeurs numériques aux coordonnées libres.

L'idée est de représenter des individus appartenant au concept Point et d'associer des coordonnées dans un espace $\mathbb{R}^n$ à chacun de ces points. Ces coordonnées sont initialement libres, puis contraintes par l'ensemble des autres assertions. Pour définir trois points alignés sur une droite nous pouvons écrire `line(p1,p2,p3)`. Comme la logique descriptive est un fragment de la logique du premier ordre à deux variables, certaines réécritures syntaxiques sont nécessaires. Pour obtenir une écriture à deux variables, il faut créer un individu appartenant au concept Line et écrire `line(l1,p1)`, `line(l2,p1)`, `line(l3,p1)`. Comme nous utilisons l'hypothèse du monde ouvert, il faut utiliser `same(p1,p2)`, `diff(p1,p2)` pour expliciter que `p1` et `p2` sont différents ou égaux. Nous pouvons mettre aussi des contraintes sur les points `>=x(p1,x)`, `>=y(p1,y)`, `>=xy(p1,x,y)`, `>=(p1,p2)` pour les contraindre à avoir des positions en x et/ou en y supérieures ou inférieures à une valeur ou bien d'avoir des coordonnées supérieures ou inférieures à celles d'un autre point. `symmetry(p1,p2,p3,p4)` définit deux points p3, p4 symétriques par rapport à la droite passant par p1, p2. `circle(p1,p2,p3)` définit trois point sur un cercle. `circle-r(p1,p2,p3)` définit deux points `p2` et `p3`, sur un cercle de centre p1. `mid(p1,p2,p3)` définit un point p3 au milieu du segment p1,p2.

Certains ensembles d'assertions géométriques n'ont pas de solutions. D'autres en ont une infinité. Les points, et les figures qu'ils composent, sont libres si aucune contraintes n'a été explicitement formulée.

Exemple : Le rectangle d'Or

Le rectangle d'or peut être représenté par un ensemble de prédicats. Il faut pour cela suivre la même logique que celle utilisée lors de la construction avec la règle et le compas. On peut alors déterminer que si l'ensemble de prédicats suivants est vrai
`carre(p1,p2,p3,p4)`
`mid(p1,p2,p5)`
`circle-r(p5,p3,p6)`
`line(p1,p2,p6)`
`perpendi(p1,p2,p6,p7)`
`line(p3,p4,p7)`
alors le prédicat `rectangleOr(p1,p6,p7,p4)` définit un rectangle d'or.

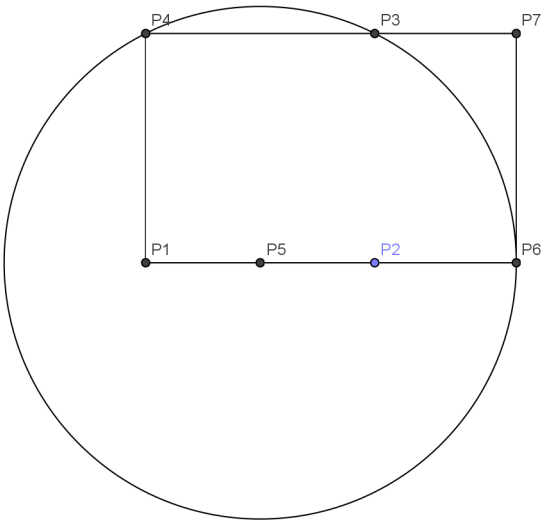


FIGURE 3.4 – Rectangle d'Or

Application d'algorithmes externes

Les logiciels suivants peuvent, par exemple, être utilisés comme algorithmes externes :

- Le logiciel FRONTIER[72], sous license GNU, est un moteur de contraintes géométriques basé sur Maple. Il permet de résoudre des contraintes géométriques entre des objets géométriques basées sur des équations algébriques. Ces équations algébriques peuvent contraindre des objets géométriques à être perpendiculaires, tangents, incidents... ou bien à respecter certaines tolérances sur des distances ou des angles. Certaines adaptations sont nécessaires pour utiliser ce logiciel avec les dernières versions de Java et de Maple.
- Le logiciel GEOCODE[2] (www.geocode.org) est quant à lui basé sur la résolution de contraintes sur des domaines finis et est applicable dans le cas discret.

3.6 Abduction

L'abduction, comme décrit dans la section 2.5, est un mécanisme d'inférence qui permet de déterminer la liste des propriétés et des relations qu'il faut nécessairement ajouter à un individu pour qu'il soit classifié dans une

classe donnée. L'idéal est de pouvoir déterminer un ensemble de propriétés et de relations non triviales et minimales.

Abduction d'un élément d'architecture dans un style donné

Par exemple, si nous voulons qu'une porte appartienne à un style donné, l'algorithme d'abduction permet de déterminer quelles sont les relations et les propriétés qu'il faut nécessairement ajouter pour que cet élément soit classifié dans ce style. S'il est nécessaire, pour cela, que la porte soit en face d'une fenêtre, l'algorithme peut proposer d'associer la porte à une fenêtre déjà existante, ou bien si nécessaire proposer la création d'une nouvelle fenêtre. Cependant, les algorithmes abductifs actuels ne permettent que des raisonnements de type monotonique. C'est-à-dire qu'il ne peuvent déterminer que les relations et propriétés qu'il faut ajouter, et non pas les éléments et relations qu'il faudrait supprimer.

3.6.1 Application de l'abduction à l'architecture

On définit tout d'abord les maisons de style X comme étant « les maisons contenant une enfilade de trois pièces dont les pièces aux extrémités sont une salle de bain et une chambre à coucher, la pièces du milieu étant pièce tampon ». En appliquant le moteur d'abduction avec seulement une chambre à coucher et une salle de bain alors le raisonneur arrive à :

- créer un nouvel individu,
- dire que c'est une pièce tampon,
- dire que cette pièce appartient à la maison,
- la connecter d'un coté avec la salle de bain et de l'autre coté avec la chambre à coucher.

En utilisant nRQL,le langage de Racer, le problème peut être exprimé de la manière suivante :

```
(full-reset)
(in-knowledge-base house)
(define-primitive-role is-connected)
(define-primitive-role belongs-to)
(disjoint house room)
(disjoint bed-room bath-room)
(implies bed-room room)
(implies bath-room room)
(implies buffer-room room)
(define-rule
   ($?h style-x)
   (and ($?h house)
        ($?x1 bed-room) ($?x2 buffer-room) ($?x3 bath-room)
```

```
        ($?x1 $?x2 is-connected) ($?x2 $?x3 is-connected)
        ($?x1 $?h belongs-to) ($?x2 $?h belongs-to)
        ($?x3 $?h belongs-to)))
(instance r1 bed-room)
(instance r3 bath-room)
(instance hh house)
(related r1 hh belongs-to)
(related r3 hh belongs-to)
(retrieve-with-explanation
   () (hh style-x) :final-consistency-checking-p t)
```

Les rôles définis sont $isConnected$ et $belongsTo$. Les concepts définis sont $house$, $styleX$, $room$, $bedRoom$, $bathRoom$ et $bufferRoom$. Les trois concepts $bedRoom$, $bathRoom$ et $bufferRoom$ sont des sous-concepts de $room$. La ABox $\mathcal{A}$ est définie comme suit :

$$\begin{aligned}\mathcal{A} \;=\; & \{bedRoom(r_1), bathRoom(r_3), house(hh),\\ & belongsTo(r_1, hh), belongTo(r_3, hh)\}\end{aligned} \tag{3.1}$$

On définit la règle suivante :

$$\begin{aligned}&house(?h), bedRoom(?x_1), bufferRoom(?x_2), bathRoom(?x_3),\\ &isConnected(x_1, x_2), isConnected(x_2, x_3),\\ &belongsTo(?x_1, ?h), belongsTo(?x_2, ?h), belongsTo(?x_3, ?h) \rightarrow styleX(?h)\end{aligned} \tag{3.2}$$

Avec l'aide du raisonneur on détermine l'ensemble d'assertions suivant :

$$\begin{aligned}\mathcal{A}_{sol} = \{&bufferRoom(newInd), belongsTo(newInd, hh),\\ &isConnected(r_1, newInd), isConnected(newInd, r_3)\}\end{aligned} \tag{3.3}$$

On aboutit ainsi, par abduction, à la création d'une nouvelle pièce qui permet de classifier la maison dans le style X.

3.7 Règles et architecture

L'utilisation de règles dans les bases de connaissance permet à la fois d'augmenter l'expressivité du formalisme, et de faciliter l'expression de certains axiomes. Les systèmes de règles peuvent être vus comme un sous-ensemble de la logique du premier ordre (Horn Logic, Definite Logic Programs)[8]. Une règle a comme forme,

$$A_1, ..., A_n \rightarrow B$$

où A_i et B sont des formules atomiques. Il y a deux principales manières d'interpréter les règles. La manière déductive : si les A_i sont connues comme

vraies alors B est vraie ; et la manière réactive : si les conditions A_i sont vraies alors exécuter l'action B.

L'utilisation d'un langage de règles sur les graphes d'une ABox ou d'une TBox permet de :

- détecter des chaines et des cycles,
- assérer l'appartenance d'un nœud à un concept,
- assérer l'appartenance d'une arrête à un rôle,
- détecter la violation d'une contrainte.

3.7.1 Règles et consistance

L'application d'une règle peut être vue comme l'adjonction d'un graphe à la base de connaissance quand une certaine configuration est détectée. Cette configuration peut être détectée en utilisant des critères purement géométriques, mais il est aussi possible d'utiliser des critères terminologiques, qui sont associés soit à la forme dans son entièreté, soit à des parties de la forme. La représentation spatiale est ainsi liée à la représentation terminologique par le biais d'une fonction de correspondance. L'application de règles peut entrainer l'inconsistance de la base de connaissance par rapport à certains axiomes ou contraintes précédemment définies. Il est donc nécessaire d'appliquer un algorithme de vérification de consistance après l'application des règles, et le cas échéant, d'annuler leur application. On peut démontrer qu'un ensemble de règles garantit forcément la consistance, mais un utilisateur classique ne dispose pas forcément des connaissances mathématiques nécessaires. Un ensemble de règles peut être appliqué de manière concurrente. Dans ce cas, pour garantir que l'application des règles engendre toujours le même résultat, il est nécessaire de définir un ordre de précédence dans l'exécution des différentes règles.

3.7.2 Règles sur des graphes d'éléments

Une règle est déclenchée en détectant un sous-graphe dans le graphe des éléments d'architecture. Il est ainsi facile de détecter tous les éléments en bois qui touchent un élément métallique, tous les éléments d'un prix supérieur à 1000 francs qui ont une fonction décorative, toutes les portes intérieures qui ont un poids supérieur à 50 kg... Lorsqu'une telle configuration est détectée, il est possible de substituer une partie du graphe par une autre, par exemple d'utiliser un élément décoratif dont le prix est de 500 francs, ou bien de rajouter un élément nouveau comme par exemple un joint entre l'élément en bois et l'élément en métal, ou bien de simplement avertir l'utilisateur par un message disant qu'il existe une contrainte sur le poids des portes.

3.7.3 Règles de classification

Les règles logiques peuvent être utilisées pour attribuer des instances à des classes. Par exemple, pour classer les objets dans leurs classes d'équivalence, nous utilisons la règle 3.5. Le classement se fera en fonction des comparaisons déjà présentes dans la base de connaissance. Ces objets peuvent être connotés avec les concepts architecturaux présents dans la branche *connotation* du module dédié à la terminologie de l'architecture.

$$
\begin{array}{lcl}
Comparateur & \sqsubseteq & \exists hasDistanceMax.int \sqcap \\
 & & \exists hasDistanceMin.int \sqcap \\
 & & \exists hasReferent.Objet \\
ComparateurSchemaCompo & \sqsubseteq & Comparateur \\
Comparateur & \equiv & ComparateurSchemaCompo \sqcup \\
 & & ComparateurTrame \sqcup \\
 & & ComparateurTrame \\
Comparaison & \sqsubseteq & \exists hasOp1.SchemaCompo \sqcap \\
 & & \exists hasOp2.SchemaCompo \sqcap \\
 & & \exists hasComparateur.Comparateur \\
ClassEquivalence & \sqsubseteq & \exists hasComparateur.Comparateur \\
Objet & \sqsubseteq & \forall inClassEq.ClassEquivalence \\
Style & \sqsubseteq & \exists hasClassEq.ClasseEquivalence
\end{array}
\tag{3.4}
$$

$$
\begin{array}{l}
Objet(?obj1) \wedge Objet(?ref) \\
\wedge ClassEquivalence(?clsEq) \wedge Comparateur(?comparateur) \\
\wedge Comparaison(?comparaison) \wedge hasComparateur(?clsEq, ?comparateur) \\
\wedge hasReferent(?clsEq, ?ref) \wedge hasDistanceMax(?comparateur, val1) \\
\wedge hasDistanceMin(?comparateur, val2) \wedge hasOp1(?comparaison, ?obj1) \\
\wedge hasOp2(?comparaison, ?ref) \wedge hasComparateur(?comparaison, ?comparateur) \\
\longrightarrow inClassEq(?obj, ?clsEq)
\end{array}
\tag{3.5}
$$

3.7.4 Exemples d'utilisation des règles

Les exemples suivants illustrent différentes possibilités d'utilisation des règles.

– Vérification de contraintes : Toutes les fenêtres doivent avoir la même hauteur. Si des hauteurs différentes sont détectées alors il faut ajouter

un axiome associant l'individu appartenant au concept $Erreur$ au concept $DesignErrorNotAllWindowsSame$. La règle se formalise de la façon suivante :

$$\begin{array}{l} Fenetre(?f1) \wedge Fenetre(?f2) \\ \wedge hasHauteur(?f1, ?h1) \wedge hasHauteur(?f2, ?h2) \wedge \\ diffAs(?h1, ?h2) \wedge Erreur(?e) \\ \rightarrow DesignErrorNotAllWindowsSame(?e) \end{array} \tag{3.6}$$

– Classer toutes les portes précédées d'une marche de plus de 10 centimètres comme étant non accessible aux fauteuils roulants.

$$\begin{array}{l} Porte(?p) \wedge Marche(?m) \wedge hasHauteur(?m, ?h) \wedge \geq (?h, 10) \\ \rightarrow NonAccesibleFauteuil(?p) \end{array} \tag{3.7}$$

– Détecter la succession d'éléments Pilier, Fenêtre, Pilier, puis créer un individu appartenant au concept $PilierFenetrePilier$ reliant les trois individus le composant. Pour cela il faut travailler en monde fermé, en utilisant un algorithme externe permettant d'ajouter un nouvel individu dans la ABox. La consistance de la base de connaissance pouvant être rompue par l'algorithme, il faudra appliquer le moteur d'inférence après la terminaison de l'algorithme.
– Pour supprimer l'appartenance d'un individu à un concept ou l'appartenance d'une relation à un rôle, il faut aussi utiliser l'hypothèse du monde fermé et un algorithme externe.

Chapitre 4

Eléments de conception architecturale

Que cela soit de manière implicite ou explicite, les outils d'aide à la conception architecturale (CAAD) sous-tendent une certaine manière de concevoir l'architecture. Le type d'outils CAAD défini ici a pour but d'expliciter par le biais d'une base de connaissance le modèle de conception architecturale sous-jacent.

Dans un premier temps, nous avons choisi, en nous basant sur les travaux d'Emmanuelle P. Jeanneret[52], d'utiliser un modèle de conception qui explicite les éléments de conception suivants : une trame, un Plus Grand Commun Descripteur, un Schéma de Composition, des éléments d'architecture paramétrables. Ces éléments de conception peuvent être modélisés par le biais des formalismes du chapitre 2 et mis en relation dans une base de connaissance. Les méthodes de raisonnement comme la classification d'instances, la subsomption, l'application de règles et l'abduction peuvent être utilisées pour la vérification des contraintes et la proposition de solutions nouvelles. Les moteurs de raisonnement comme Pellet[89], Racer[43], ou HermiT[68], peuvent être utilisés pour ce faire.

Les éléments de conception décrits dans ce chapitre permettent de modéliser toutes une variétés de styles architecturaux différents. Ils ne sont cependant pas suffisant pour modéliser l'ensemble des styles imaginables. Il s'agit d'une des limitations de ce travail. Il est toutefois possible en gardant la même structure d'ajouter de nouveaux éléments de conception. Si nécessaire, il est aussi possible d'ajouter de nouveau formalismes à condition que ceux-ci puissent s'interconnecter avec la logique descriptive.

4.1 La trame

La trame sous-tend les lignes principales d'un édifice. Elle partitionne le plan en des régions appelées cellules, qui peuvent être de taille et de forme semblables ou différentes. Une trame est propre à un style architectural donné. Elle est mathématiquement déterminée par deux ensembles de courbes. Ces deux ensembles doivent respecter les conditions suivantes :

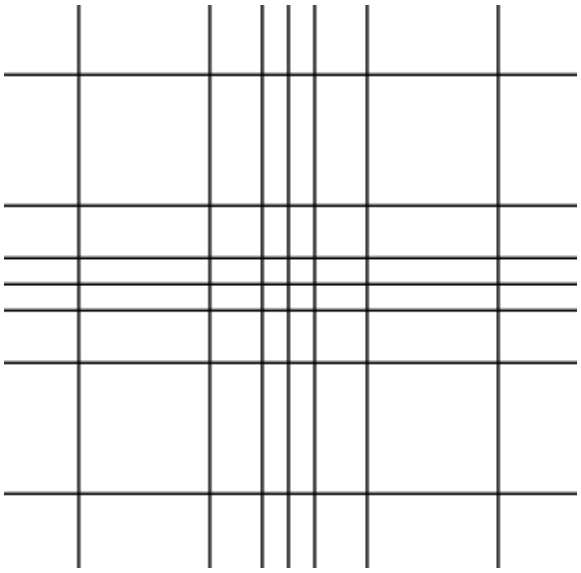


FIGURE 4.1 – Trame respectant la progression de Fibonacci

- les courbes appartenant à un même ensemble ne doivent pas s'intersecter,
- chaque courbe d'un ensemble doit s'intersecter une et une seule fois avec chaque courbe de l'autre ensemble.

Les autres éléments de conception comme le Plus Grand Commun Descripteur ou le Schéma de Composition sont dessinés sur la trame. Les éléments de surface délimités par la trame seront ensuite utilisés dans les différentes opérations de composition.

4.2 Le plus grand commun descripteur

Le plus grand commun descripteur (PGCD) est une forme géométrique simple qui caractérise la conception d'un ensemble édifices ou d'un style. Le PGCD est dessiné en respectant les lignes de la trame. En appliquant selon

des règles données un ensemble d'opérations géométriques sur cette forme, on peut construire le Schéma de Composition d'un édifice particulier.

4.3 Le schéma de composition

Le schéma de composition est, comme le PGCD, une forme dessinée sur la trame. Il donne les lignes principales d'un édifice et permet de guider le placement des éléments d'architecture. C'est généralement une forme au contour fermé qui peut posséder des frontières intérieures. La construction de ce schéma se fait par l'application d'une succession d'opérations géométriques (translation, rotation, symétrie, homothétie, dilatation,...) sur une forme initiale. Ces opérations peuvent aussi être appliquées à une sous-partie d'une forme. Différentes méthodes permettent d'encoder le schéma de composition. Il est possible, entre autres, d'utiliser une forme initiale et une suite d'opérations géométriques ; une représentation vectorielle sous la forme d'une suite de vecteurs ; une composition de carrés ; ou bien encore une description des contours extérieurs et intérieurs.

4.4 Les éléments d'architecture paramétrables

Les éléments d'architecture sont les éléments concrets qui s'inscrivent dans l'espace. Ils possèdent un ensemble de propriétés et sont définis géométriquement. Les éléments d'architecture sont liés les uns aux autres par des relations à la fois dans l'espace et dans la structure logique du projet.

4.5 Tâches utilisateurs

La conception d'un projet architectural par l'utilisateur consiste, en terme de modèles décrits dans ce chapitre, à déterminer une Trame, un Schéma de Composition ainsi qu'un ensemble d'Eléments d'architecture positionnés sur ce schéma. Les contraintes et les objectifs fixés par l'utilisateur, les éléments déjà définis ainsi que la base de connaissance, fournissent les matériaux nécessaire à l'application des tâches de raisonnement.

Un langage de requêtes permet au concepteur d'interroger la base de connaissance qui contient des éléments de projets antérieurs ainsi qu'une théorie générale de l'architecture. Les éléments qui sont les plus proches de sa problématique actuelle peuvent être consultés et réutilisés. De plus, des

moteurs de raisonnement déductifs et abductifs sont à sa disposition pour lui permettre d'obtenir des nouvelles solutions partielles.

Les fonctionnalités d'un système d'aide à la conception architecturale basé sur les connaissances peuvent être les suivantes :

- construire et/ou modifier une base de connaissance de l'architecture,
- construire et/ou modifier un projet architectural basé sur les trames, les schémas de composition et les éléments d'architecture,
- respecter les contraintes d'un PGCD
- formuler des contraintes et des objectifs par le biais du langage de requêtes,
- vérifier la conformité du projet par rapport aux contraintes et objectifs spécifiés,
- obtenir une assistance automatique permettant de résoudre les contradictions entre l'état du projet, les contraintes et les objectifs,
- classifier les éléments du projet par rapport à une ontologie générale de l'architecture ou à une ontologie spécifique à un style donné,
- réutiliser des éléments de projet stockés dans la base de connaissance dans son propre projet,
- rechercher les éléments de la base de connaissance proches des éléments du projet en cours,
- appliquer des règles de construction automatiques,
- obtenir la liste des actions nécessaires pour atteindre un objectif donné.

4.5.1 Langage de requêtes spécifique à l'architecture

En utilisant les formalismes décrits précédemment, il est possible de définir un langage de requête qui combine les éléments suivants :

- Des **requêtes en logique descriptive**
  qui permettent de déterminer :
  - l'ensemble des instances présentes dans la base de connaissance qui appartiennent à un concept donné, comme par exemple toutes les portes en bois dans un mur porteur :

$$Porte \sqcap \forall hasMatière.Bois \sqcap \exists estPartieDe.MurPorteur$$

  - l'ensemble des sous-concepts d'un concept donné. Par exemple tous les sous-concept de $EspaceFonctionel$.
  - l'ensemble des super-concepts d'un concept donné. Par exemple tous les super-concepts de $PorteEnBois$.
- Des **requêtes en logique spatio-terminologique (avec logique topologique)**

qui permettent de déterminer l'ensemble des instances présentes dans la base de connaissance qui appartiennent à un concept donné et qui sont inscrites dans un réseau de relations spatiales spécifiées de manière plus ou moins fortes. Pour cela on peut, par exemple, utiliser Pellet Spatial comme décrit à la section 3.4.1. En utilisant ce type de requêtes, il est possible de déterminer, par exemple, tous les objets inflammables qui touchent des cheminées.

– Des **requêtes en logique spatio-terminologique (avec logique métrique)**
qui permettent d'effectuer des requêtes incluant des contraintes de distance entre les objets spatiaux. A chaque objet du domaine spatial peut être associé une position. Avec l'aide d'un algorithme externe, il est ensuite aisé de calculer toutes les distances nécessaires et d'évaluer des prédicats de distance du type $\delta(x, y) \leq 5$. La combinaison de la logique descriptive et de la logique métrique peut se faire avec une logique du type de $\mathcal{ALC}(\mathcal{MS})$ comme défini à la section 2.7.6.

– Des **requêtes par règles SWRL**
qui permettent de générer des nouveaux axiomes à inclure dans la base de connaissance. On peut ainsi détecter les économies d'énergie possible :

$$\begin{aligned} &EspaceChaud(?x) \wedge EspaceFroid(?y) \wedge ElementArchi(?z) \wedge \\ &\quad NonIsolant(?z) \wedge touche(?x, ?z) \wedge touche(?z, ?y) \\ &\quad \rightarrow ElementARemplacerPourEconomie(?z) \end{aligned} \tag{4.1}$$

– Des **requêtes en SPARQL-DL**
qui permettent d'utiliser des variables pour les concepts et pour les rôles. Par exemple quels sont les types de fenêtre posées dans les murs porteurs :

$Type(?f; Fenêtre); Type(?f; ?C); MurPorteur(?m); hasPart(?m; ?f)$

– Des **requêtes par le biais d'algorithmes externes**
qui permettent d'effectuer des opérations non réalisables en logique. Par exemple, le calcul de taux sur des graphes avec le logiciel GED. Une fois calculé, les résultats des algorithmes sont réinjectés dans la base de connaissance. Il est ensuite nécessaire de vérifier que la base de connaissance est toujours consistante d'un point de vue logique.

4.5.2 Distances entre les graphes de relations

La définition d'une mesure de distance entre des éléments de conception offre la possibilité d'obtenir, à partir d'une requête, les instances les plus

proches d'une instance choisie. Chaque instance ou concept en logique descriptive peut être associé à un graphe décrivant ses relations. Dans le cas d'une instance, chaque instance liée transitivement à l'instance que l'on veut représenter est représentée par un nœud. En d'autres termes, il s'agit de sélectionner le sous graphe connexe auquel l'individu à représenter fait partie. Un exemple d'un tel graphe pour une instance de porte est donné à la figure 4.2. Dans le cas d'un concept, il s'agit de déterminer un sous-graphe

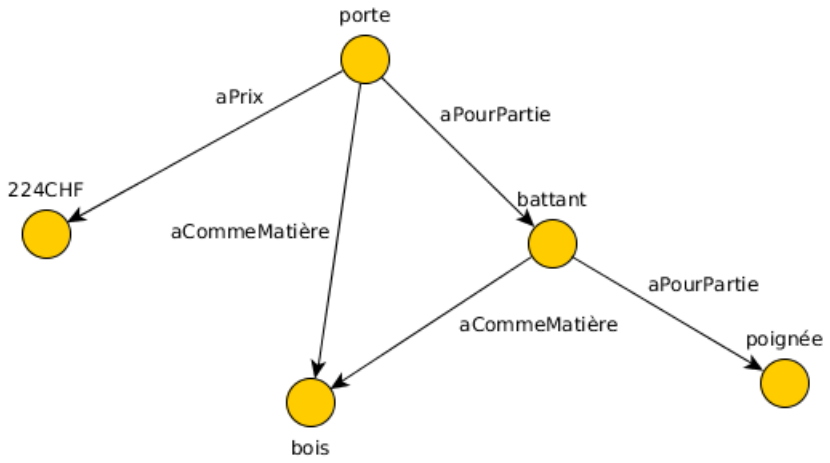


FIGURE 4.2 – Graphe d'une instance de porte

connexe de la TBox. Par exemple, pour le concept

$$Piece \sqcap \exists hasPart.Ouverture \forall \sqcap hasMur.MurEnBrique$$

le graphe correspondant est donnée par la figure 4.3. Il est ensuite possible de calculer des distances entre ces différents graphes par le biais d'algorithmes appropriés.[25]. Les distances dynamiques, basées sur le nombre d'opérations pour passer d'un graphe à un autre ainsi que les distances statiques, basées sur des similitude de structure, permettent d'obtenir les instances et les concepts apparentés entre eux. Cependant, pour obtenir une distance sémantique, la distance entre graphes ne suffit pas, puisqu'il faut prendre en compte des opérateurs sémantiques comme la négation[50].
De cette manière, il est possible d'effectuer des requêtes basées sur des mesures de similarité (distance). On peut, par exemple, chercher dans un catalogue tous les escaliers les plus proches d'un escalier donné. En fonction

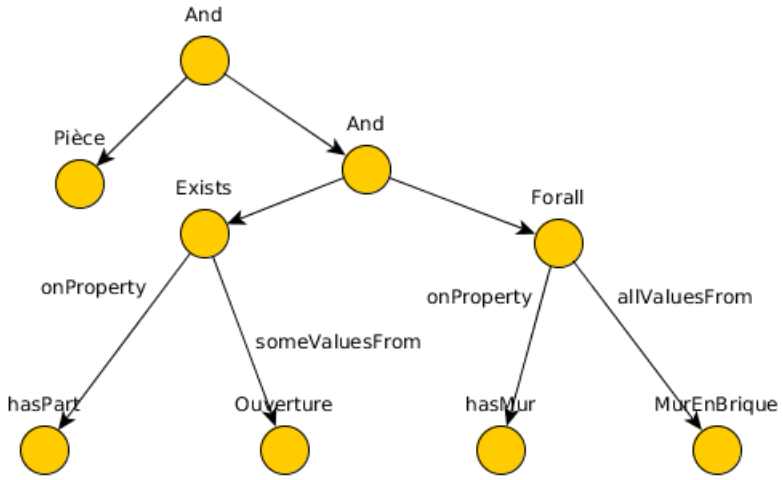


FIGURE 4.3 – Graphe d'un concept de pièce

de la modélisation effectuée, l'algorithme de distance peut être appliqué sur des graphes de concepts ou bien sur des graphes d'instances.

4.5.3 Vérification des contraintes

Un ensemble de contraintes sur une ABox représentant un projet en cours de conception peut être exprimé selon différents formalismes :

- Des **contraintes en logique descriptive** sous la forme d'axiomes dont la consistance avec un ensemble d'assertions logiques est vérifiable par des raisonneurs comme Pellet ou HermiT. Par exemple, un axiome imposant qu'un éléments architectural ne puisse pas être à la fois une ouverture et une fermeture est exprimé par :

  $$(ElementArchi \sqcap Ouverture) \sqcap (ElementArchi \sqcap Fermeture) \equiv \bot$$

  ce qui peut être réécrit sous la forme

  $$ElementArchi \sqcap Ouverture \sqcap Fermeture \equiv \bot$$

- Des **contraintes par des règles SWRL** qui permettent de vérifier qu'il n'existe pas de configurations d'instances et de relations interdites entre les individus nommés. Par exemple, une règle vérifiant que deux

murs non porteurs ne doivent pas être accolés, est exprimée par :

$$(\neg MurPorteur)(?m1) \wedge (\neg MurPorteur)(?m2) \wedge touche(?m1, ?m2) \wedge Status(?s) \rightarrow ErreurMurPorteur(?s) \tag{4.2}$$

- Des **contraintes en RCC8** qui vérifient la consistance des relations spatiales de type topologiques par l'algorithme de consistance par chemins[83]. Par exemple si nous avons trois pièces p_1, p_2, p_3 et que ces pièces sont liées par les relations topologiques $PO(p_1, p_2), PO(p_2, p_3), NTPP(p_1, p_2)$ alors ces relations sont détectées comme inconsistantes par l'algorithme.
- Des **contraintes définies par des algorithmes externes** Un algorithme externe peut tester si la structure de la base de connaissance vérifie certains invariants. Par exemple, l'ensemble des murs appartenant à une pièces doivent constituer un cycle fermé. Un algorithme externe peut vérifier cette contraintes en faisant l'hypothèse du monde fermé, à savoir que tous les murs de la pièce sont déjà connu.

4.5.4 Classification et recherche d'instances similaires

Les différents individus stockés dans une base de connaissance peuvent être classifiés dans une ou plusieurs classes de l'ontologie. A partir d'un individu donné, il est possible de retrouver tous les individus de la base de connaissance qui sont classés dans des classes identiques. Il est aussi possible, à partir d'un individu classé dans une classe C, de retrouver tous les individus qui sont classés dans des classes subsumées ou subsumant la classe C.

Recherche d'éléments d'architecture appartenant à la même classe

Si dans un projet on utilise une porte-fenêtre, il est possible de lister l'ensemble des autres instances de portes-fenêtres qui sont utilisées dans le même style.

Recherche d'éléments d'architecture appartenant à un concept similaire

A partir de la description logique d'un concept, il est possible de relâcher certaines contraintes. Si par exemple aucune fenêtre similaire à une fenêtre d'un style donné n'est présente dans la base de connaissance, on peut relâcher les contraintes spécifiques aux matériaux et ainsi obtenir la liste des instances de fenêtre proche du style recherché, mais dans des matériaux différents.

Recherche d'instances similaires par le biais d'algorithmes externes

Suivant le type d'instances, il est possible d'utiliser des algorithmes externes afin de déterminer des instances similaires. Pour des formes par exemple, un ensemble de valeurs basées sur des propriétés géométriques peuvent être stockées dans un vecteur. Il est ensuite possible d'utiliser des distances dans cet espace de paramètres. Ces distances peuvent ensuite être réimportées dans la base de connaissance et combinées à des expressions logiques.

Chapitre 5

Modélisation des éléments de conception architecturale

Dans ce chapitre, les différents éléments de conception architecturale définis dans le chapitre 4 seront modélisés afin de permettre l'application d'algorithmes et de tâches de raisonnement utiles au concepteur d'un projet architectural. Une logique métrique permettant d'associer des points géométriques à des instances de logique descriptive sera utilisée pour modéliser la trame. Les schémas de composition seront modélisés par des liste d'opérations, des ensembles de segments, ou bien par des ensembles de pièces. Les représentations par instances ainsi que les représentations par concepts seront étudiées. Des algorithmes externes aux modèles logiques seront utilisés afin de contourner les limitations de ceux-ci. On verra aussi comment modéliser des éléments d'architecture et les positionner dans l'espace en s'aidant des schémas de composition.
Le modèle comporte les limitations suivantes :

- il ne peut pas prendre en compte l'évolution du projet à travers le temps, *logiques temporelles*,
- il ne permet pas de supprimer des éléments lors des raisonnements, *logiques non-monotoniques*,
- il ne permet pas de traiter des appartenance à des classes ou des relations entre individus de manière probabiliste, *logiques floues*.

5.1 Modélisation de la trame

Une trame est déterminée par deux ensembles de courbes qui respectent les conditions données dans la section 4.1. A savoir, chaque courbe d'un ensemble doit croiser exactement une fois toutes les courbes de l'autre ensemble.

Une trame constituée par des droites peut être modélisée par une suite de distances entre les lignes horizontales et une suite de distances pour les lignes verticales.

En utilisant la logique $\mathcal{ALC}(\mathcal{MS})$ (c.f. 2.7.6), chaque nœud de la trame peut être associé à la fois à un point du plan et à un individu de logique descriptive. On utilise les nominaux $n_0, n_1, n2, ...$ de la logique $\mathcal{MS}$, pour décrire des points géométriques sur la trame. On utilise la fonction d'extension conceptuelle ς^- pour associer des individus $c_0, c_1, c_2, ...$ de la logique $\mathcal{ALC}$ à des points géométriques dénommés par les nominaux $n_0, n_1, n2, ...$ De manière réciproque la fonction d'extension spatiale ς permet d'associer des points géométriques de la trame aux individus de la logique $\mathcal{ALC}$ représentant ces nœuds. Les droites de la trame sont définies par les variables d'ensembles X_i et Y_j. Le nœuds à l'intersection de ces droites est décrit par $X_i \cap Y_j$.

Si plusieurs trames sont superposées, on peut associer aux points géométriques des nœuds des individus appartenant au concept $Trame_1$ ou $Trame_2$ respectivement. Un même nœud peut appartenir à plusieurs trames si l'individu correspondant appartient au concept $Trame_1 \sqcap Trame_2 \sqcap ....$

On peut ainsi, par exemple, définir le concept contenant l'ensemble des individus situés à une distance de moins de 5, ou égale à 5, d'un individu nœuds c_0 par $\varsigma^-(E_{\leq 5}\varsigma(c_0))$ et le concept contenant l'ensemble des individus situés à une distance de plus de 10 des individus constituant la trame 3 par $\varsigma^-(E_{\geq 10}\varsigma(Trame_3))$.

5.2 Modélisation du schéma de composition

Le Schéma de Composition peut être représenté de différentes manières. Les représentations utilisées ici sont :

- une liste d'opérations géométriques appliquée à une forme initiale,
- un ensemble de segments positionnés sur la trame,
- un ensemble de pièces définies par leur contour et liées les unes aux autres par des relations de connexités.

Les principaux concepts logiques utilisés sont les suivants :

- *Cell* : unité de surface du plan
- *Segment* : segment séparant deux unités de surface
- *Room* : ensemble d'unités de surfaces connexes dont l'enveloppe définit une pièce

5.2.1 Modélisation par une liste d'opérations

Une forme géométrique peut être représentée par la suite des opérations géométriques nécessaires à sa construction. Le type d'opérations et les manières admises de composer ces opérations définissent une classe de formes donnée. Dans la section 6.2, nous avons, sur le conseil des architectes et selon l'histoire des pratiques architecturales, appliqué des successions de translations unitaires et demi unitaires sur un carré initial.
Plusieurs suites d'opérations différentes permettent de construire une même forme. Ainsi, en utilisant une de ces suite d'opérations pour représenter une forme, nous obtenons une représentation qui n'est pas univoque. La figure

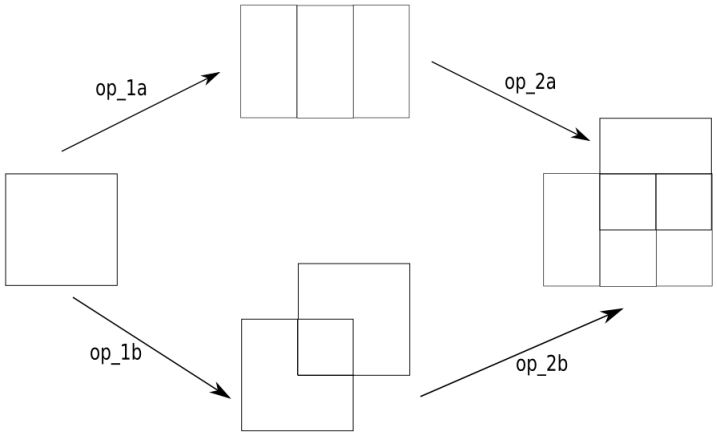


FIGURE 5.1 – Graphe des opérations permettant la construction d'une forme

5.1 montre comment deux suites de translations différentes permettent de construire une même forme. Chaque chemin eulerien du graphe (chaque arrête est parcourue exactement une fois) de la figure 5.1 détermine une suite d'opérations permettant la construction de la forme finale.
Ces opérations sont stockées sous la forme de concepts dans la base de connaissance et une forme géométrique est modélisée par une liste OWL (comme dans la section 5.3.3) d'opérations géométriques.

5.2.2 Modélisation par un ensemble de segments

Chaque élément unitaire de la trame est appelé cellule. A chaque cellule de la trame sont associés de zéro à quatre segments de courbe. Les cellules forment un pavage du plan. Quatre relations de positions sont possibles entre deux cellules voisines. Ces quatre relations correspondent aux quatre points

cardinaux (Nord, Sud, Est, Ouest). La modélisation par ensembles de segments permet de stocker un schéma de composition sous la forme d'une ABox et ainsi d'appliquer les différentes méthodes d'inférence de la logique descriptive sur ces schémas de composition.
Pour un pavage du plan issu d'une trame telle que définie dans la section 4.1, les rôles DL suivants sont utilisés pour définir les positions relatives entre les cellules et entre les pièces (ensemble de cellules connexes) :

- N l'individu i_1 est au Nord de i_2,
- W l'individu i_1 est à l'Ouest de i_2,
- E l'individu i_1 est à l'Est de i_2,
- S l'individu i_1 est au Sud de i_2.

On notera respectivement N_R, N_C ou N_S lorsque l'on voudra distinguer un rôle entre des pièces, un rôle entre des cellules ou un rôle entre une cellule et un segment. Ces trois rôles étant subsumés par le rôle N, à savoir $N_R \sqsubseteq N$, $N_C \sqsubseteq N$ et $N_S \sqsubseteq N$.
Les axiomes suivants peuvent être définis :

- Les rôles N et S sont des rôles inverses :
  $i_1 N i_2$ est équivalent à $i_2 S i_1$.
- Les rôles W et E sont des rôles inverses :
  $i_1 W i_2$ est équivalent à $i_2 E i_1$.
- Les rôles N_C, W_C, E_C, S_C sont fonctionnels :
  Si $i_1 N_C i_2$ alors i_1 ne peut pas être lié par la relation N_C à un autre individu.
- Les rôles N_C, W_C, E_C, S_C sont inversement fonctionnels :
  Si $i_1 N_C i_2$ alors i_2 ne peut pas être lié par la relation N_C à un autre individu.
- Les rôles N, W, E, S sont asymétriques :
  Si $i_1 N i_2$ alors il est impossible que $i_2 N i_1$.
- Les rôles N, W, E, S sont irréflexifs :
  Il est impossible que $i_1 N i_1$.

Un individu appartenant au concept *Cell* est créé pour chaque unité du plan. L'ensemble des cellules sont stockées dans la ABox. Chaque cellule est reliée à ses quatre voisines par les relations N_C, W_C, E_C, S_C. Les segments sont représentés par des individus appartenant au concept *Segment*. Les cellules sont liées aux segments qui les entourent par les rôles N_S, W_S, E_S, S_S.

Les schémas de composition peuvent être décrits soit par des instances soit par des concepts. Un schéma de la ABox sera classé dans tous les concepts de schémas qui le contiennent au sens géométrique.

Pour traiter plusieurs Schémas de Composition dans une même ABox, on utilise le concept *SchemaComposition* et les rôles *hasPart* et $hasPart^{-1}$. Chaque cellule, mur et pièce peut ainsi appartenir à plusieurs schémas de

composition. Il faudra néanmoins vérifier la consistance du modèle, en vérifiant que les individus liés entre eux appartiennent au même schéma.

Exemple de représentation d'un schéma de composition par instances

On représente la forme de la figure de 5.2a par un ensemble d'individus et de relations entre ces individus. On utilise 4 individus appartenant au concept $Cell$ et 12 individus appartenant au concept $Segment$. Tout d'abord, on lie les cellules entre elles. Il est possible d'utiliser uniquement les rôles S_C et E_C dans la ABox, car les assertions contenant les rôles N_C et W_C peuvent être déduites en utilisant les axiomes suivants de la TBox : $N_C \equiv S_C^{-1}$ et $W_C \equiv E_C^{-1}$. La ABox $\mathcal{A}_0$ contient uniquement les relations concernant les cellules :

$$\begin{aligned}\mathcal{A}_0 = \{&Cell(c_0), Cell(c_1), Cell(c_2), Cell(c_3), Cell(c_4), Cell(c_5),\\ &E_C(c_0, c_1), E_C(c_2, c_3), E_C(c_4, c_5), S_C(c_0, c_2), S_C(c_2, c_4)\}\end{aligned} \tag{5.1}$$

On ajoute ensuite les segments, avec $Segment \equiv Seg$:

$$\begin{aligned}\mathcal{A}_1 = \mathcal{A}_0 \cup \{&Seg(s_0), Seg(s_1), Seg(s_2), Seg(s_3), Seg(s_4), Seg(s_5),\\ &Seg(s_6), Seg(s_7), Seg(s_8), Seg(s_9), Seg(s_{10}), Seg(s_{11})\}\end{aligned} \tag{5.2}$$

On associe les cellules aux segments :

$$\begin{aligned}\mathcal{A}_2 = \mathcal{A}_1 \cup \{&N_S(c_1, s_0), W_S(c_1, s1), E_S(c_1, s_2), S_S(c_1, s_4),\\ &N_S(c_2, s_3), W_S(c_2, s_5), S_S(c_2, s_7),\\ &E_S(c_3, s_6), S_S(c_3, s_8), W_S(c_4, s_9), E_S(c_4, s_{10}), S_S(c_4, s_{11})\}\end{aligned} \tag{5.3}$$

Dans $\mathcal{A}_2$ chaque segment appartient à une seule cellule. Cependant chaque segment est à la frontière entre deux cellules. Il faudra donc compléter la ABox, lors du raisonnement, en utilisant les quatre règles suivantes :

$$Seg(?s_0) \wedge Cell(?c_0) \wedge Cell(?c_1) \wedge N_S(?c_0, ?s_0) \wedge N_C(?c_0, ?c_1) \rightarrow S_S(?c_1, ?s_0)$$

$$Seg(?s_0) \wedge Cell(?c_0) \wedge Cell(?c_1) \wedge W_S(?c_0, ?s_0) \wedge W_C(?c_0, ?c_1) \rightarrow E_S(?c_1, ?s_0)$$

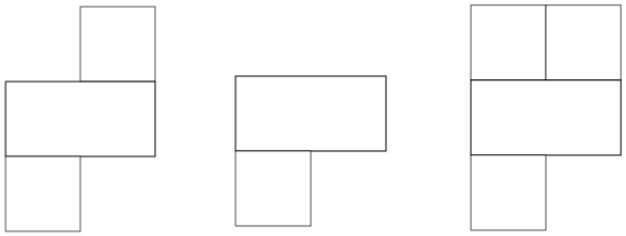
$$Seg(?s_0) \wedge Cell(?c_0) \wedge Cell(?c_1) \wedge E_S(?c_0, ?s_0) \wedge E_C(?c_0, ?c_1) \rightarrow W_S(?c_1, ?s_0)$$

$$Seg(?s_0) \wedge Cell(?c_0) \wedge Cell(?c_1) \wedge S_S(?c_0, ?s_0) \wedge S_C(?c_0, ?c_1) \rightarrow N_S(?c_1, ?s_0)$$

En appliquant ces règles, la ABox est complétée de telle sorte que chaque segment appartient a exactement deux cellules.

Application de la représentation par concepts

Les descriptions de concepts peuvent être utilisées pour décrire des classes de Schémas de Composition. Si on définit un schéma de composition par un sous-ensemble de segments $Seg(\mathcal{S})$ d'une trame, une classe de Schémas de Composition $\bar{\mathcal{S}}_c$ est définie par l'ensemble de segments nécessairement présents $Seg(\bar{\mathcal{S}}_c)$ sur cette trame. Un Schéma de Composition appartient à une classe de schémas de composition ssi $Seg(\mathcal{S}) \subseteq Seg(\bar{\mathcal{S}}_c)$. Tous les schémas de composition qui incluent le schéma $\mathcal{S}_c$ appartiennent à la même classe d'équivalence notée $\bar{\mathcal{S}}_c$.



(a) $Shape_a \in ABox$ (b) $Shape_b \in ABox$ (c) $Shape_c \in TBox$

FIGURE 5.2 – Forme en DL

Un schéma de composition $\mathcal{S}$, représenté par une ABox $\mathcal{A}$ est indépendant de sa position sur la trame. Par contre, dans le cas de la représentation par concepts, une classe de schémas de composition est définie par rapport à un point de référence, que l'on situe en haut à gauche du rectangle englobant de la classe de schémas.

Pour construire le concept qui représente la classe de schémas de composition, il faut donc déterminer le rectangle englobant la classe de schémas de composition. Puis, partant de la cellule en haut à gauche de ce rectangle, il faut parcourir l'ensemble des cellules, ligne par ligne, de gauche à droite. Le concept correspondant à une cellule entourée de quatre murs se décrit comme suit :

$$\text{Carré} \equiv \text{Cell} \sqcap \exists N.\text{Seg} \sqcap \exists W.\text{Seg} \sqcap \exists E.\text{Seg} \sqcap \exists S.\text{Seg} \tag{5.4}$$

Le concept correspondant à une classe de schémas de composition contenant

la forme de la figure 5.2c est le suivant :

$$
\begin{array}{l}
\text{Forme} \equiv \\
\text{Cell} \sqcap \exists N.\text{Seg} \sqcap \exists W.\text{Seg} \sqcap \exists E.\text{Seg} \sqcap \exists S.\text{Seg} \\
\sqcap \exists E.(\text{Cell} \sqcap N.\text{Seg} \sqcap \exists W.\text{Seg} \sqcap \exists E.\text{Seg} \sqcap \exists S.\text{Seg}) \\
\sqcap \exists S.(\text{Cell} \sqcap \exists N.\text{Seg} \sqcap \exists W.\text{Seg} \sqcap \exists S.\text{Seg} \\
\sqcap \exists E.(\text{Cell} \sqcap \exists N.\text{Seg} \sqcap \exists E.\text{Seg} \sqcap \exists S.\text{Seg}) \\
\sqcap \exists S.(\text{Cell} \sqcap \exists N.\text{Seg} \sqcap \exists W.\text{Seg} \sqcap \exists E.\text{Seg} \sqcap \exists S.\text{Seg} \sqcap \exists E.\text{Cell})
\end{array}
\tag{5.5}
$$

La représentation de schémas de composition par concepts permet d'utiliser la subsomption pour vérifier si une classe de schémas de composition est incluse dans une autre.

Une classe de schémas de composition $\bar{\mathcal{S}}_a$ est incluse dans une autre de ces classes $\bar{\mathcal{S}}_b$ si les segments nécessairement présents dans la classe $\bar{\mathcal{S}}_a$ sont nécessairement présents dans la classe $\bar{\mathcal{S}}_b$ à savoir $Seg(\bar{\mathcal{S}}_a) \subseteq Seg(\bar{\mathcal{S}}_b)$.

De manière plus générale, en définissant une application ς qui associe à chaque classe de schéma de composition $\bar{\mathcal{S}}_c$ un concept S_c, on a :

$$
Seg(\bar{\mathcal{S}}_a) \subseteq Seg(\bar{\mathcal{S}}_b) \qquad \text{ssi} \qquad \varsigma(Seg(\bar{\mathcal{S}}_b)) \sqsubseteq \varsigma(Seg(\bar{\mathcal{S}}_a)) \tag{5.6}
$$

On peut ainsi vérifier que le concept Carré défini par l'équation 5.4 est subsumé par le concept Forme définie par l'équation 5.5. Cela indique que la classe de schémas de composition décrite par le concept Carré est incluse dans la classe de schémas de composition décrite par le concept Forme.

A partir d'un ensemble de schémas de composition représentés par des concepts, on peut construire un treillis entre le concept $\top$ et le concept $\bot$ décrivant les différentes inclusions entre les classes de schémas de composition.

Application de la représentation par instances

Dans le cas de la représentation par instances, il est possible de détecter une sous-forme, soit par le biais de la vérification d'instances (instance checking), soit par le biais de l'utilisation de règles. Le schéma de composition décrit par instances dans l'équation 5.3 est une instance du schéma de composition décrit sous forme de concept par l'équation 5.5. En utilisant les règles, il est en outre possible de déterminer le nombre d'occurrences d'une sous-forme dans une forme.

Annotation des schémas de composition par des concepts d'architecture

L'avantage d'utiliser une représentation logique pour les formes apparait quand il s'agit d'annoter les différentes parties de la forme avec des descriptions de concepts. Ces descriptions peuvent contenir tous les concepts présents dans la terminologie. Les descriptions de parties de formes peuvent même contenir d'autres parties de formes.
Tout concept décrivant une partie de la forme est subsumé par le concept *Description*. La description d'une porte-fenêtre est notée $Porte \sqcap Fenetre \sqsubseteq Description$. Les constituants de la forme peuvent ainsi être liés à une description par le rôle *hasDescription*. Une même description peut être utilisée à différents endroits et une même partie de forme peut posséder plusieurs descriptions.

5.2.3 Unique Name Assumption

Le formalisme de la logique descriptive est basé sur l'hypothèse non-UNA (non-Unique Name Assumption). Ainsi un même individu peut posséder plusieurs noms. De la même manière deux individus de noms différents peuvent être identiques. Des axiomes supplémentaires sont nécessaires pour lever l'ambigüité. Ces axiomes sont de la forme *SameAs(i1,i2)* pour indiquer que deux individus sont identiques ou *DifferentFrom(i1,i2)* pour indiquer que deux individus sont différents.

La représentation par instances de la forme de la figure 5.2a nécessite la présence de 12 segments. Chaque segment appartient à deux cellules placées à la verticale l'une de l'autre ou à l'horizontale l'une de l'autre. A la création d'une cellule, il est plus aisé de définir les segments qu'elle possède plutôt que de chercher les segments préexistants dans les cellules voisines.

Il est ainsi possible de créer pour chaque cellule les segments qui lui sont propres puis de déterminer par le moyen des deux règles suivantes les segments qui sont communs entre les cellules. Ces deux règles, exprimées avec le formalisme SWRL, permettent de compléter les axiomes d'identité :

$$\begin{aligned} & Cell(?c1) \wedge Cell(?c2) \\ & \wedge N_C(?c1, ?c2) \\ & \wedge Segment(?s1) \wedge Segment(?s2) \\ & \wedge N_S(?c1, ?s1) \wedge S_S(?c1, ?s2) \\ & \rightarrow SameAs(?s1, ?s2) \end{aligned} \tag{5.7}$$

$$
\begin{aligned}
&Cell(?c1) \wedge Cell(?c2) \\
&\wedge E_C(?c1, ?c2) \\
&\wedge Segment(?s1) \wedge Segment(?s2) \\
&\wedge E_S(?c1, ?s1) \wedge W_S(?c1, ?s2) \\
&\rightarrow SameAs(?s1, ?s2)
\end{aligned} \tag{5.8}
$$

5.2.4 Passage des segments aux murs

Un segment est défini comme une séparation entre deux cellules. Un mur est quant à lui défini comme une suite connexe de segments orientés dans la même direction. Les règles logiques ne sont pas un formalisme adéquat en ce qui concerne la détection d'une suite d'un nombre inconnu de segments. En effet, la longueur d'un mur étant inconnue, il faudrait donc une règle par longueur de mur pour pouvoir effectuer la détection. L'utilisation d'un algorithme externe est ici nécessaire. Cet algorithme ne pourra opérer que sur les segments explicitement présents dans la base de connaissance. L'hypothèse du monde fermé sera donc utilisée.

5.2.5 Représentation d'une forme par des murs

Il est possible de décrire une forme non plus avec des cellules et des segments mais uniquement avec des murs. On procède de la manière suivante :

Chaque mur possède deux extrémités et peut être placé dans deux orientations possibles : verticale ou horizontale. Pour la suite nous faisons l'hypothèse que la première extrémité est toujours à gauche dans le cas des murs horizontaux et toujours en haut dans le cas des murs verticaux. La relation la plus évidente entre deux murs est qu'ils soient déconnectés. Si deux murs sont dans la même orientation et se touchent alors ils se suivent. Les rôles $isFollowedBy$ et $isFollowedBy^{-1}$ sont utilisés pour décrire ces relations. Si deux murs sont dans des orientations différentes et se touchent alors il y a quatre relations possibles entre ses murs. Par convention, on décide d'utiliser le sens horaire. Un segment horizontal se parcoure de gauche à droite et un segment vertical se parcoure du haut vers le bas. On définit les relations, relatives à l'orientation, $cornerLeft$, $cornerRight$, $cornerLeft^{-1}$ et $cornerRight^{-1}$. Les sept relations précédemment décrites sont fonctionnelles, inversement fonctionnelles, asymétriques et irréflexives. Les murs verticaux $WallV$ et les murs horizontaux ne peuvent pas être combinés de manière arbitraire. On ajoute à la base de connaissance les axiomes suivants :

$$WallV \sqcap \exists isFollowedBy.WallH \equiv \bot$$

$$WallV \sqcap \exists cornerLeft.WallV \equiv \bot$$

$$WallV \sqcap \exists cornerRight.WallV \equiv \bot$$

Les trois axiomes correspondant pour $WallH$ pouvant facilement être déduit. Pour dessiner une forme à partir d'une ABox contenant des murs, il faut appliquer un algorithme de la forme Breadth-first search (BFS). Pour que l'algorithme fonctionne il faut que l'orientation des murs soit définie, c'est-à-dire que les individus murs appartiennent à $(MurV \sqcap \neg MurH) \sqcup (MurH \sqcap \neg MurV)$. Certains murs dont l'orientation n'est pas déterminée peuvent être orientés par l'application de règles comme :

$$MurV(?m_0) \wedge Mur(?m_1) \wedge isFollowedBy(?m_0, ?m_2) \rightarrow MurV(?m_1)$$

5.2.6 Relation entre pièces et murs

A chaque pièce est associée un ensemble de murs par le rôle $hasWall$. A chaque mur est associé au plus deux pièces $inRoom$.

$$Room \sqsubseteq \forall hasWall.Wall$$

$$Wall \sqsubseteq \leq 2\ inRoom.Room$$

5.2.7 Détection et relations entre pièces

La figure 5.3 obtenue par une suite d'opérations sur une forme simple est décrite en logique descriptive de la même manière que la forme de la figure 5.2a. En utilisant un algorithme récursif, il est possible d'associer à chacune des surfaces fermées un label. Le concept Room décrit précédemment contient un ensemble d'individus qui sont reliés par le rôle hasCell aux différents individus du concept Cell. Les différents individus Room sont reliés par les rôles N, W, E, S. Cependant ces rôles, dans le cas de relations entre pièces, ne satisfont plus les mêmes axiomes. Ils ne sont plus ni fonctionnels, ni inversement fonctionnels, ni asymétriques. La pièce intérieure de la figure 5.4 est liée par les rôles N, W et S à la pièce gauche et par les relations N, E, S à la pièce droite. Ces relations ne peuvent donc plus être fonctionnelles. De plus la pièce gauche a une relation N avec la pièce intérieure et la pièce intérieure a elle-même une relation N avec la pièce gauche. Les relations ne peuvent donc, non plus, être asymétriques.
Deux pièces connexes ont en commun un ensemble de murs.

5.2.8 Représentation canonique d'une pièce

La forme d'une pièce peut être représentée de manière canonique en décrivant son enveloppe extérieure. On définit tout d'abord le point le plus

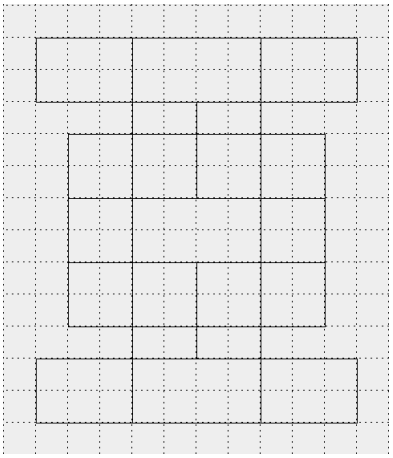


FIGURE 5.3 – Forme en DL

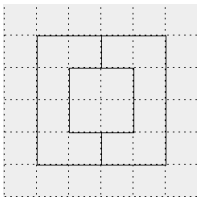


FIGURE 5.4 – Forme en DL

en haut à gauche de l'enveloppe. On parcourt ensuite l'enveloppe dans le sens horaire, en décrivant à chaque étape si l'on avance, si l'on tourne à gauche ou si l'on tourne à droite. Une enveloppe peut être ainsi définie comme une succession de murs. Cette ensemble de murs doit satisfaire un ensemble de contraintes :

- (1) Chaque mur doit posséder un et un seul successeur appartenant à l'enveloppe de la pièce.
- (2) L'enveloppe de la pièce doit être fermée.
- (3) L'enveloppe de la pièce ne doit pas s'intersecter elle même.

La contrainte (1) peut être définie en logique du premier ordre par l'expression

$$\forall w_1 \in Wall, \exists! w_2 \in Wall, \exists e \in Envelop(hasPart(e, w_1) \wedge hasPart(e, w_2) \wedge hasNext(w_1, w_2))$$

$\exists!$ étant le quantificateur d'unicité défini par

$$\exists! x P(x) \equiv \neg \exists x, y (P(x) \wedge P(y) \wedge (x \neq y))$$

Il n'est pas possible d'exprimer la contrainte (1) en logique descriptive ni en SWRL car il n'y a pas de *negation as failure* dans ce langage. Il n'est pas non

plus possible d'exprimer cette contrainte par le biais de SPARQL-DL, il s'agit en effet de détecter tous les cycles de longueur quelconque et de vérifier que pour chaque cycle l'ensemble des arrêtes appartiennent à la même enveloppe. Par contre l'algorithme suivant permet dans le cas du monde fermé de vérifier si la contrainte est respectée.

```
Pour chaque mur m
   Pour chaque enveloppe E associé au mur m
      Si m n'est pas traité pour l'enveloppe E
         marquer m comme traité pour l'enveloppe E
         marquer m comme initial
         mcourant = m
         Faire
            Si il n'y a pas de suivant pour l'enveloppe E
               return false
            Si il y a plus que un suivant du même type que mcourant
               return false
               mcourant = suivant
               marqué mcourant comme traité pour le type T
            Tant que mcourant n'est pas initial
               marquer mcourant comme non-initial
   Fin Pour Chaque
Fin Pour Chaque
return true
```

L'algorithme peut être modifié pour vérifier que dans le cas du monde ouvert la contrainte (1) n'est pas violée en prenant en compte les connaissances actuelles.

```
Pour chaque mur m
   Pour chaque type T associé au mur m
      Si m n'est pas traité pour le type T
         marquer m comme traité pour le type T
         marquer m comme initial
         mcourant = m
         Faire
            Si il n'y a pas de suivant de type T
               break
            Si il y a plus que un suivant du même type que mcourant
               return false
               mcourant = suivant
               marqué mcourant comme traité pour le type T
            Tant que mcourant n'est pas initial
        marquer mcourant comme non-initial
   Fin Pour Chaque
Fin Pour Chaque
return true
```

Les contraintes (2) et (3) ne peuvent non plus être définies en logique et doivent être vérifiées par des algorithmes.

La pièce ainsi définie possède un certain nombre de propriétés déductibles de l'enveloppe de la pièce.

- un périmètre,
- une surface,
- un rectangle englobant
- des moments d'inerties

A partir d'un ensemble arbitraire de murs, nous pouvons donc vérifier s'il forme une pièce, le transformer en une représentation canonique, extraire des propriétés comme la surface, le périmètre, lui appliquer des opérations comme la symétrie, la rotation.

5.2.9 Propriétés d'un Schéma de Composition

Différents types de propriété peuvent être extraites d'un Schéma de Composition. Il est possible de déterminer sa surface, son périmètre, son facteur de forme, ses moments d'inertie,... Il est aussi possible d'extraire le graphe des relations entre les pièces, les murs, les cellules et pour chacun de ces graphes de calculer un ensemble de propriétés (notamment avec le logiciel GED[26]). Ces propriétés sont calculées à partir de la ABox par des algorithmes spécifiques. Une fois calculées, elle peuvent être réinjectées dans la ABox et utilisées dans les raisonnements logiques. Cependant, à chaque modification d'un schéma de composition dans la ABox, il faudra recalculer ces propriétés pour qu'elles soient à jour. Afin d'optimiser le temps de calcul, il est aussi possible de déclencher le recalcul des propriétés seulement si certaines instances de la ABox ont été modifiées.
A chaque instance de Schéma de Composition est ainsi associé un vecteur de propriétés. Les Schémas de Composition peuvent ainsi être le sujet de tous les traitement sur les *feature vector*, comme la reconnaissance de formes ou l'apprentissage automatique.

5.3 Modélisation des éléments d'architecture

Chaque élément d'architecture appartient à une ou plusieurs classes de l'ontologie. Ces classes d'éléments peuvent être définies de manière extensive en nommant explicitement un certaine nombre d'éléments. Les classes peuvent aussi être définies de manière intensive en définissant des propriétés communes à un ensemble d'éléments d'architecture, comme par exemple le fait d'être lié avec l'extérieur de l'édifice, ou d'avoir une fonction d'usage particulière. Chaque élément d'architecture est modulable par un ensemble de paramètres. L'ensemble de ces paramètres répondent à certains invari-

ants. Dans le cas d'un escalier, un des invariants est le fait que la hauteur de l'escalier doit correspondre au nombre de marches multiplié par la hauteur de ces marches. Certains de ces paramètres sont covariants entre eux. Dans l'exemple de l'escalier, une variation de la hauteur de l'escalier va entrainer une variation de la hauteur des marches, une variation du nombre de marches, ou bien une variation des deux. Les éléments d'architecture sont en relation les unes avec les autres, ils forment entre eux un graphe de relations. Ces relations peuvent définir des relations de propriété comme par exemple *aCommeMatière*, *aCommePrix*, *aCommeFonction* des relations spatiales comme par exemple *est dessus*, *touche*, *est inclus* ou bien des relations de composition comme par exemple *aCommePartie*. Un élément d'architecture peut être composé d'autres éléments d'architecture. Dans ce cas les invariants concernent l'ensemble des sous-éléments et la variation du paramètre d'un élément peut faire covarier le paramètre d'un autre élément.

5.3.1 Placement à l'intérieur d'une pièce

Les éléments d'architecture peuvent être reliés à une pièces par la relation inRoom. Une pièce peut quant à elle être reliée par la relation inverse hasEA à un ou plusieurs éléments d'architecture. Si l'on veut que le rôle hasEA soient caractérisé par un ensemble de paramètres, il est nécessaire, dans le formalisme de la DL, de créer un individu qui caractérise cette relation. Ces individus appartiennent au concept EAlink.

$$\text{EA} \sqsubseteq \exists \text{inRoom.Room} \sqcap \forall \text{inRoom.Room} \tag{5.9}$$

$$\text{EA} \sqsubseteq \exists \text{hasLink.}(\text{EAlink} \sqcap \exists \text{hasLink.Room}) \tag{5.10}$$

$$\text{inRoom} \equiv \text{hasLink} \circ \text{hasLink} \tag{5.11}$$

L'équation 5.9 définit comme nécessaire qu'un élément d'architecture soit lié à au moins une pièce. L'équation 5.10 définit comme nécessaire qu'un élément d'architecture soit lié par un individu du concept EAlink à une pièce. Quant à l'équation 5.11 elle définit le rôle inRoom comme la composition de deux rôles hasLink. On peut maintenant associer des paramètres à un individu EAlink :

$$\text{EAlink} \sqsubseteq \exists alignementX.Integer \sqcap \exists alignementY.Integer \tag{5.12}$$

Les individus EAlink doivent ainsi posséder deux valeurs d'alignement entières.

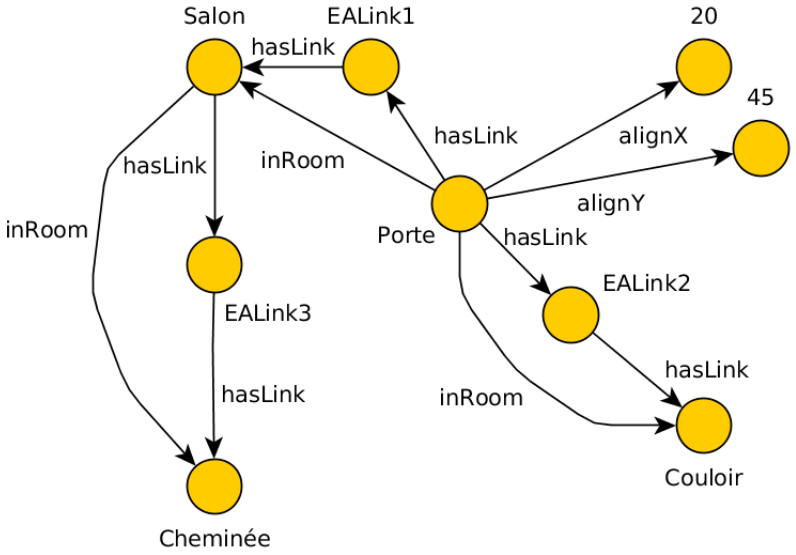


FIGURE 5.5 – Graphe d'instances des relations entre pièces et éléments d'architecture

5.3.2 Façades et éléments d'architecture

Dans cette section, les éléments d'architecture seront composés entre eux dans des façades. Ces façades seront connectées entre elles par leurs extrémités. Le terme *façade* est défini ici comme une succession d'éléments en série. Une façade peut être, par exemple, constituée de :

- un mur plein
- une succession de mur-fenêtre-mur-fenêtre-...
- un mur-fenêtre-mur-porte-mur-fenêtre-mur
- une succession de pilier-vide-pilier-vide-...
- un vide

La description de ces façades en logique descriptive est faite dans la section 5.3.3 et dans la section 5.3.4. Chaque façade est munie d'un paramètre décrivant sa longueur totale et d'un autre décrivant son orientation dans le plan.

A chaque façade sont associées deux extrémités. Ces extrémités sont reliées entre elle par des arcs annotés par un vecteur de déplacement $(\Delta x, \Delta y)$ et par un sous-concept du concept façade issu de la base de connaissance. Ces arcs peuvent ensuite être représentés dans une matrice d'incidence associant les différentes extrémités entre elles. Comme les façades ne peuvent pas

s'intersecter, cette matrice doit correspondre à un graphe planaire. Prenons comme exemple un simple carré. Chacune des extrémités est reliée par un

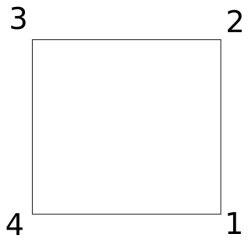


FIGURE 5.6 – Carré munis de quatre extrémité

arc au deux autre extrémités, pour l'instant les arcs ne sont pas annotés par des concepts mais seulement par un vecteur $(\Delta x, \Delta y)$. La matrice suivante décrit ces relations :

$$\begin{pmatrix} \emptyset & (0,1) & \emptyset & (-1,0) \\ (0,-1) & \emptyset & (-1,0) & \emptyset \\ \emptyset & (1,0) & \emptyset & (0,-1) \\ (1,0) & \emptyset & (0,1) & \emptyset \end{pmatrix} \tag{5.13}$$

La diagonale est composée d'ensembles vide ce qui indique que les extrémités ne sont pas connectées à elles-même. La première ligne indique que :

- il n'y a pas de relations entre l'extrémité 1 et l'extrémité 1,
- l'extrémité 1 est liée à l'extrémité 2 avec un déplacement de $\Delta x = 0$ et $\Delta y = 1$,
- il n'y a pas de relations entre l'extrémité 1 et l'extrémité 3,
- l'extrémité 1 est liée à l'extrémité 4 avec un déplacement de $\Delta x = -1$ et $\Delta y = 0$.

Avec cette notation, il est possible de décrire n'importe quel tracé de graphe. Le forme de la figure 5.7 est ainsi représentable dans ce formalisme. Pour que

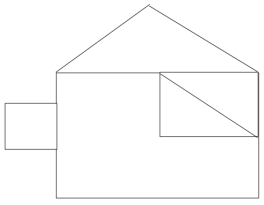


FIGURE 5.7 – Forme représentable

la matrice d'incidence représente un tracé correct, il faut que, pour chaque cycle du graphe, la somme des Δx et la somme des Δy soient nulles.

Une autre manière de représenter un plan architectural par un graphe est d'annoter non plus les arcs mais les sommets. Chaque sommet est ainsi annoté par une position (x, y) dans le plan. Pour passer d'une représentation à une autre il suffit de parcourir le graphe avec un algorithme de type BFS (Breadth First Search), en assignant une position arbitraire au premier sommet rencontré. Le passage d'une représentation par position à une représentation par déplacement se fait de manière similaire.

A une étape donnée du processus de conception, il est possible de définir certaines positions et certains déplacements de manière seulement partielle. Pour ce faire, les positions et les déplacements peuvent être définis par des plages de valeurs admissible. Prenons la forme de la figure 5.8. Le point 1

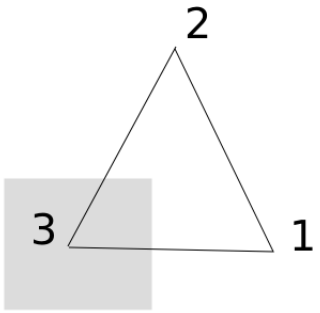


FIGURE 5.8 – Forme partiellement définie

est situé à une position fixe. Le point 2 a une coordonnée en x fixe et une coordonnée en y libre. La position du point 3 est quant à elle contrainte dans le carré gris. De plus, on ajoute la contrainte $x_1 + y_2 < x_3$. Par des méthodes de programmation linéaire, il est possible de calculer l'ensemble des formes admissible.

5.3.3 Modélisation par listes

Les listes d'éléments d'architecture permettent de représenter des successions d'éléments d'architecture présents, par exemple sur des façades, ou le long de chemins. Chaque élément de la liste appartient à un sous-concept du concept *ElementArchi*. En logique descriptive, on peut représenter la liste

en utilisant des concepts et des rôles de la manière suivante[23] :

$$\begin{array}{lcl} OWLList & \sqsubseteq & \exists isFollowedBy.OWLList \\ EmptyList & \sqsubseteq & OWLList \\ EmptyList & \sqsubseteq & \leq 0\ hasContent \\ EmptyList & \sqsubseteq & \neg\exists hasNext.\top \\ hasNext & \sqsubseteq & isFollowedBy \end{array} \tag{5.14}$$

Les relations $hasNext$ et $hasContent$ sont fonctionnelles. La relation $isFollowedBy$ est transitive et a comme rang $OWLList$. Une liste contenant un mur, une porte et un mur est exprimée de la façon suivante :

$$\begin{array}{l} OWLList \sqcap \exists hasContents.Mur \sqcap \exists hasNext.(\\ OWLList \sqcap \exists hasContents.Porte \sqcap \exists hasNext.(\\ OWLList \sqcap \exists hasContents.Mur \sqcap \exists hasNext.EmptyList)) \end{array} \tag{5.15}$$

A chacun des éléments de la façade est associé un coefficient d'élasticité compris entre 0 et 1 borne incluse. La somme des coefficients devant être égal à 1. Ainsi, si la longueur totale de la façade est incrémentée d'une valeur Δx, cette différence de longueur est répartie sur les différents éléments en fonction de leurs coefficient d'élasticité.

5.3.4 Modélisation par cycles

Un plan architectural peut-être décomposé en un ensemble de cycles élémentaires. Ces cycles sont obtenus en parcourant pour chacune des pièces la suite des murs la composant. Les cycles sont composés d'une suite de façades ou d'éléments d'architecture. Dans ce cas, la relation $isFollowedBy$ n'a plus de sens puisque chaque élément est suivi de tous les autres. Un cycle composé de quatre murs et d'une porte est défini de la manière suivante :

$$\begin{array}{l} MemberCycle_1 \equiv Porte \sqcap \exists Next. \\ (Mur \sqcap \exists hasNext. \\ (Mur \sqcap \exists hasNext. \\ (Mur \sqcap \exists hasNext. \\ (Mur \sqcap \exists hasNext.MemberCycle_1)))) \end{array} \tag{5.16}$$

De cette manière, un cycle est une liste commençant par un individu donné et finissant par ce même élément. Un même cycle peut cependant être représenté de plusieurs manières différentes en fonction de l'individu qui est choisi pour débuter le cycle. Si nous prenons l'exemple le plus simple d'un cycle de deux

éléments, appartenant respectivement aux concepts C_1 et C_2, on définit les membres qui sont dans le cycle et qui appartiennent au concept C_1 par :

$$\begin{array}{l} MemberCycle_1 \equiv C_1 \sqcap \exists Next.(\\ C_2 \sqcap \exists hasNext.MemberCycle_1) \end{array} \tag{5.17}$$

et les membres qui sont dans le cycle et qui appartiennent au concept C_2 par :

$$\begin{array}{l} MemberCycle_2 \equiv C_2 \sqcap \exists Next.(\\ C_1 \sqcap \exists hasNext.MemberCycle_2) \end{array} \tag{5.18}$$

Pour définir un membre de ce cycle de manière univoque, il faut prendre la disjonction de ces deux définitions :

$$MemberCycle \equiv MemberCycle_1 \sqcup MemberCycle_2 \tag{5.19}$$

5.3.5 Positionnement et paramétrisation des éléments d'architecture

Le positionnement et la paramétrisation se fait selon un ensemble de règles :

- des règles de symétries : Si une règle de symétrie n'est pas respectée alors un individu permettant de rétablir la symétrie est proposé par un algorithme d'abduction.
- des règles de présence nécessaire : Si une certaine configuration est détectée et qu'elle nécessite la présence d'élément ou de relation absents alors l'élément ou la relation en question est proposé par un algorithme d'abduction.
- des règles de paramètres admissibles : Si une configuration est détectée et qu'elle nécessite qu'un ou plusieurs individus soient liés à des paramètres de certaines valeurs alors ces paramètres sont proposés à l'utilisateur.
- des règles de relations entre paramètres : Si une configuration est détectée et que les relations entre certains paramètres ne sont pas respectées alors un ensemble de paramètres acceptable est proposé.
- des règles de respect de positions : Si une configuration est détectée et que les positions de différents élément ne sont pas acceptables alors un ensemble de positions acceptables est proposé.

L'ensemble des paramètres liés à un élément d'architecture doit respecter un ensemble d'*invariants*. En d'autres termes, certaines propriétés globales doivent être respectées. L'ensemble des paramètres liés à un élément d'architecture est partitionné en des sous ensembles de *covariants*. Modifier un

seul des éléments d'un ensemble de covariants nécessite de modifier les autres éléments de cet ensemble.
Les éléments d'architecture peuvent aussi être composés d'autres éléments d'architecture. Les invariants et les covariants d'un élément peuvent relier des sous-éléments d'architecture composant un même élément.

5.4 Structure de la base de connaissance

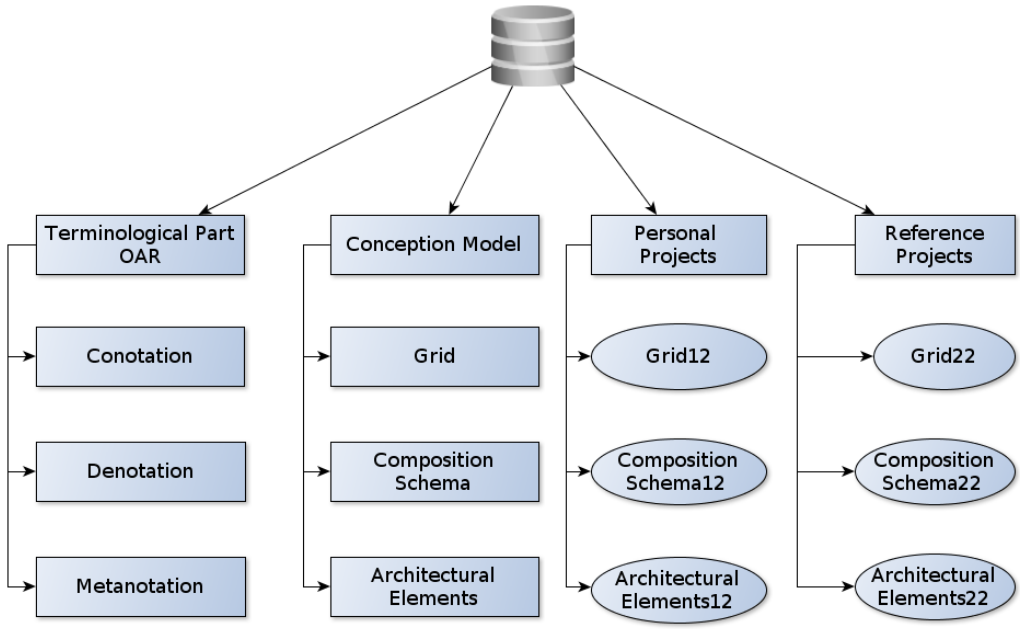


FIGURE 5.9 – Vue schématique de la base de connaissance

La base de connaissance est composée d'un ensemble de modules. Chaque type d'éléments de conception abstraits, comme par exemple les Trames ou les Schémas de Composition, sont associés à un module particulier de la base de connaissance. Un module spécifique dédié à la terminologie de l'architecture a été développé par nos collègues architectes (OAR[77]). Il faut en outre distinguer la partie utilisée pour la définition des concepts et des rôles (TBox) de la partie contenant les projets du concepteur et les projets de référence (Vitruve, Palladio, Kahn, Le Corbusier).

De manière générale, on peut dire que la base de connaissance contient : des descriptions de sous-ensembles d'instances (*Concepts*), des descriptions

de propriétés communes à certains sous-ensemble d'instances (*Datatype properties*), des descriptions de relations entre des instances (*Object properties*) ainsi que des instances (*Individuals*).
La base de connaissance peut contenir plusieurs projets en cours, des projets déjà terminés ou bien des éléments provenant d'édifices de référence. Chaque trame, schéma de composition ou élément d'architecture peut être relié à un ou plusieurs projets. Les projets sont eux mêmes des instances de la base de connaissance et peuvent posséder un ensemble de propriétés et des relations. Les requêtes peuvent ainsi être effectuées sur un ou plusieurs projets.

Le module de la base de connaissance dédié aux trames contient :

- des propriétés permettant de décrire les trames (carrée, régulière, basée sur le nombre d'or, ...),
- des types de relations entre les trames ($T_1\ R\ T_2$)(basée sur les propriétés, plus ou moins fine, ...),
- des opérations sur les trames ($T_1\ R\ T_2$)(rotation, changement d'échelle,...).
- des opérations de composition entre les trames ($T_1\ R\ T_2 \rightarrow T_3$)(agrégation,...)

Le module de la base de connaissance dédié aux schémas de composition contient :

- des propriétés permettant des décrire les schémas de composition (surface, périmètre, facteur de forme, ...),
- des types de relations entre les schémas de composition (basées sur les propriétés),
- des opérations sur les schémas de composition
- des relations de composition entre schémas de composition,

Le module de la base de connaissance dédié aux éléments d'architecture contient :

- des propriétés permettant des décrire les éléments d'architecture,
- la composition des éléments d'architecture en sous-parties par le biais de relations *partOf* (méronymie),
- les paramètres définissant leurs formes et leurs métriques,
- les types d'associations entre éléments,
- les relations spatiales de connexité,
- les relations spatiales d'alignement.

La partie dédiée à la terminologie et à la théorie de l'architecture (OAR) contient :

- une branche *dénotation* qui décrit les objets constituant les édifices,
- une branche *connotation* qui inclut des spécificités propre à des styles architecturaux
- une branche *métanotation* qui permet de décrire le discours

Chapitre 6

Implémentation d'un prototype

Le prototype développé permet de tester différents formalismes, algorithmes et méthodes de raisonnements décrits dans les chapitres précédents. Certains formalismes décrit dans le chapitre 2 ne sont pas implémentés dans le prototype, c'est le cas notamment des $\mathcal{E}$-connection et de l'algèbre de directions CDC. Le système est interconnecté avec différents modules extérieurs déjà existants. Certaines parties sont réalisées en Python, d'autres dans le langage Processing[82] mais la grande majorité du prototype a été développé en Java. Les entrées sorties se font par le biais du format OWL, d'objets JAVA sérialisés ou de formats spécifiques à des modules extérieurs donnés. Un architecte utilisant ce prototype peut ainsi effectuer les tâches principales suivantes :

- dessiner des schémas de composition,
- placer des éléments d'architecture sur ces schémas et les paramétrer,
- dessiner des plans d'architecture,
- ouvrir et sauver des schémas et des plans,
- exporter et importer des schémas et des plans en OWL.

Les moteurs d'inférences et les éditeurs de bases de connaissance peuvent être utilisés sur les représentations OWL exportées. Les résultats peuvent ensuite être réimportées dans le prototype. En appliquant les procédures de raisonnement automatiques, il devient possible de :

- classifier les éléments du projet par rapport à une ontologie générale de l'architecture ou à une ontologie spécifique à un style donné,
- rechercher les éléments de la base de connaissance proches des éléments du projet en cours,
- réutiliser des éléments de projet stockés dans la base de connaissance,
- vérifier la conformité du projet par rapport aux contraintes et objectifs spécifiés,
- appliquer des règles de construction automatiques,

– créer de nouvelles instances par abduction,
– obtenir une assistance automatique permettant de résoudre les contradictions entre l'état du projet, les contraintes et les objectifs.

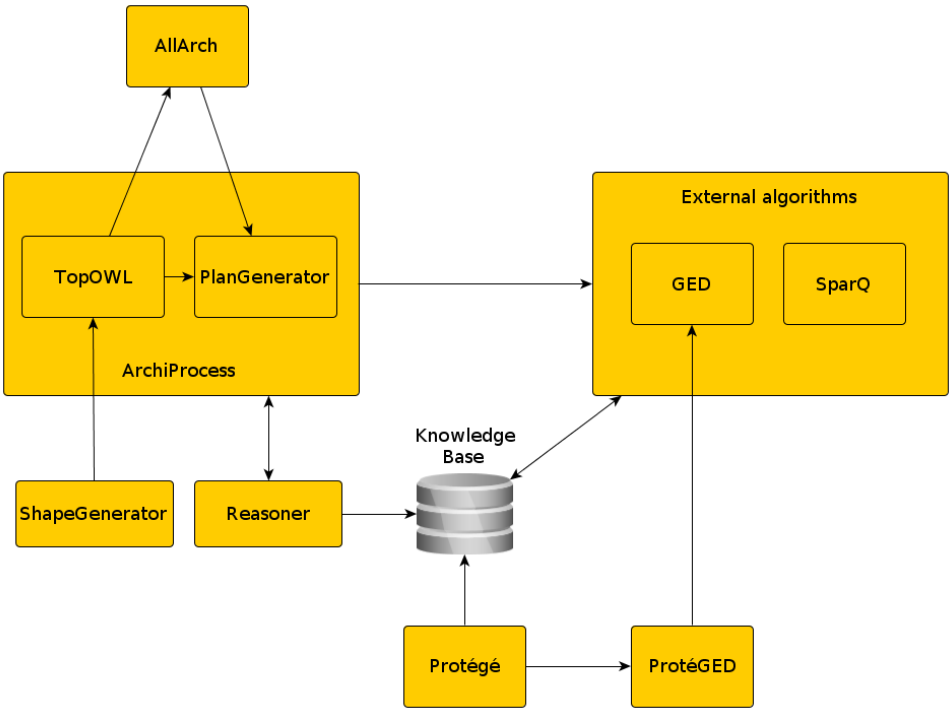


FIGURE 6.1 – Schéma d'interconnection des constituants du prototype et des outils externes

6.1 Architecture générale du prototype

La figure 6.1 décrit l'interaction entre les différents constituants du prototype et les outils externes. Le coeur du système est constitué par ArchiProcess qui est lui même constitué de deux modules : TopOWL et PlanGenerator. TopOWL est un éditeur de schéma de composition qui permet l'importation et l'exportation en logique descriptive et dans un format propriétaire qui permet la connection vers l'outils AllArch. PlanGenerator est un éditeur de plan architectural qui permet d'utiliser les schémas provenant de TopOWL et de les habiller avec des éléments d'architecture paramétrables. De la même manière que TopOWL, PlanGenerator permet l'exportation et l'importation vers la base de connaissance, il utilise aussi un format propriétaire à base

d'objets JAVA sérialisés. Le logiciel ShapeGenerator, écrit en Python, permet la génération de formes par combinaisons géométriques. L'interconnexion avec TopOWL, bien que ne représentant pas de difficultés particulières n'est pas encore réalisée. Le logiciel AllArch, cité précédemment, est un prototype réalisé par Emmanuelle P. Jeanneret, dans le cadre du projet FNS « Formalisation et sens du projet architectural ». Il permet, entre autre, le paramétrage et la composition, d'éléments d'architecture. ArchiProcess permet par le biais de la base de connaissance d'appliquer des algorithmes externes implémentés dans des logiciels comme GED ou SparQ. Il faut encore mentionné ProtéGED, un plugin Protégé, qui permet d'extraire des graphes de la base de connaissance, de calculer un ensemble de propriétés sur ces graphes puis de réintégrer ces résultats dans la base de connaissance.

6.2 ShapeGenerator

ShapeGenerator est un logiciel écrit en Python permettant de générer des formes puis de les classer selon un vecteur de caractéristiques extraites. Les formes sont générées en se basant sur une méthode issue de la théorie de l'architecture.[52]. On part d'une forme initiale qui est un carré. Les opérations permises sont des translations de une ou une demie unité. On génère ensuite toutes les suites d'opérations possibles avec un nombre d'opérations fixe. Pour chacune de ces formes un vecteur de caractéristiques géométriques est calculé. Il est ensuite possible d'effectuer des recherches dans l'ensemble des formes générées en se basant sur les vecteurs caractéristiques. L'algorithme principal est le suivant :

```
def generateSolutions(nbSquares = 5, nbOperation = 20):
   #Compute all linear combinations of nbSquares with nbOperations
   nbSolution  = nbOperation**(nbSquares-1)
   #Compute the solutions
   sol       = np.ndarray((nbSolution,nbSquares - 1),dtype=int)
   operation = 0
   for numCol in range(nbSquares - 1):
      indCol = 0
      vectorLength = nbOperation**numCol  #length of the same op vector
      vector = np.ones([vectorLength,1], dtype=int)
      for numLi in range(nbSolution/vectorLength):
         sol[indCol:indCol+vectorLength,numCol:numCol+1] = operation * vector
         operation = (operation + 1) % nbOperation
         indCol += vectorLength
      operation = 0
return sol
```

6.3 ProteGED

6.3.1 GED

GED[26] est un logiciel permettant de calculer des propriétés numériques, appelées taux, sur des graphes issus de plan d'architecture. Ces graphes peuvent définir des relations entre des pièces ou des relations entre des murs. GED accepte en entrée des plans d'architecture dessinés avec le logiciel TOP[85] et en extrait les graphes pour calculer différents taux. Il peut aussi prendre en entrée un fichier de format propriétaire décrivant directement les graphes.

6.3.2 Protégé

Protégé est une application JAVA basée sur un modèle de composants dynamiques du type OSGi[98]. Cette application permet d'éditer des bases de connaissance et d'appliquer des raisonnements automatiques.

6.3.3 Le plugin ProteGED

ProteGED est un plugin Protégé permettant d'exporter des graphes issus de la TBox ou de la ABox en format GED. Ces graphes peuvent être basés sur n'importe quel type de rôles. Il peuvent être, par exemple, basés sur le rôle $estConnectéA$ ou bien sur une hiérarchie $partOf$ et ce tant au niveau des instances que des concepts. Ce graphe est ensuite traité par le logiciel GED qui calcule un ensemble de taux, c'est-à-dire un ensemble de propriétés sur ce graphe. Ces propriétés peuvent donc ensuite être réimportées dans la ABox. GED agit donc comme un algorithme externe qui augmente la base de connaissance. Après l'augmentation de la base de connaissance, la consistance doit être vérifiée car il faut reclassifier les individus dont les propriétés ont changé. ProteGED dispose en outre d'une option permettant d'inclure ou non les sous-rôles dans le graphe généré.

6.4 AllArch et le langage Processing

Processing[82] est un langage de programmation *open source* développé au MIT. Il a pour objectif de permettre à des architectes ou à des artistes de créer des objets 2D ou 3D interactifs. Il est basé sur le langage JAVA mais son utilisation est nettement plus simple. La partie orientée objet de

JAVA est masquée, et un ensemble de primitives graphiques est directement accessible. Il dispose en outre d'un environnement de développement permettant de réaliser très rapidement des *sketch* (dessins interactifs). Processing est donc idéal pour donner la possibilité à des architectes de développer des éléments d'architecture.
Un prototype nommé *AllArch* a été développé par Emmanuelle P. Jeanneret

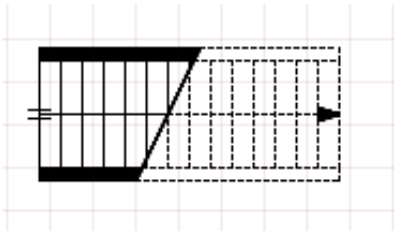


FIGURE 6.2 – Représentation d'un escalier paramétrable par le biais de AllArch

avec l'aide de l'auteur de cette thèse. Ce prototype réalisé dans le langage Processing permet, entre autres, de paramétrer et de combiner des éléments d'architecture entre eux. Les éléments d'architecture peuvent ensuite être positionné dans l'espace en se basant sur un schéma de composition provenant d'ArchiProcess.
Un élément d'architecture est ainsi défini par un ensemble de paramètres, un ensemble de relations vers d'autres éléments d'architecture comme par exemple des sous-parties ou des éléments connexes ainsi que par une méthode de dessin fonction de ces paramètres. Les paramètres définissent la forme géométrique de l'élément d'architecture ainsi que ses propriétés physiques, fonctionnelles ou autres. Les éléments d'architecture sont placés dans une hiérarchie $partOf$ ainsi que dans une hiérarchie isA. L'ensemble des propriétés définissant l'élément d'architecture et ses sous-parties doivent respecter un ensemble d'invariants spécifiques à cet élément. La vérification de ces invariants se fait lors de l'affectation d'une nouvelle valeur à un paramètre. Des paramètres covariants peuvent être modifiés lors de l'affectation afin de respecter les invariants. Par exemple, dans le cas d'un escalier, un invariant peut imposer que le nombre de marches multiplié par la hauteur d'une marche soit égale à la hauteur de l'escalier. En modifiant la hauteur de l'escalier un algorithme est déclenché calculant une nouveau nombre de marches et si nécessaire une nouvelle hauteur de marche.
Les éléments d'architecture sont classés dans une hiérarchie telle que donnée par l'ontologie de l'architecture. A chaque élément est associé un objet JAVA muni de méthodes permettant de modifier et de lire les différents paramètres.

Chaque élément est aussi lié à une classe OWL. Cette classe est liée via des *datatype property* à son ensemble de propriétés.

6.5 ArchiProcess

ArchiProcess est une application développée en JAVA. Les principales fonctionnalités implémentées sont les suivantes :

- interface de dessin de schémas de composition sur des trames,
- application d'opérations géométriques sur les schémas de composition (translation, symétries horizontales et verticales et rotations).
- agrégation des résultats des opérations avec les schémas de composition
- enregistrement des opérations géométriques qui peuvent être faites et défaites (undo/redo)
- représentation des schémas de composition en OWL sous la forme de segments,
- représentation des schémas de composition en OWL sous forme de succession d'opérations géométriques,
- placement et paramétrisation des éléments d'architecture sur un schéma de composition,
- représentation des éléments d'architecture en OWL sous la forme d'un graphe d'éléments munis de propriétés,
- exportation et importation des différentes représentations sous forme de fichiers OWL ou d'objets JAVA sérialisés.
- détection automatique des pièces présentes dans un schéma de composition.

En terme d'implémentation, une particularité d'ArchiProcess est le fait que l'interface utilisateur est construite dynamiquement en déterminant par réflexion les types de paramètres des éléments d'architecture.

6.5.1 Conversion entre éléments d'architecture en JAVA et en OWL

Les classes JAVA peuvent être représentées en OWL en faisant correspondre des DatatypeProperty aux variables d'instance de type primaire (entiers, flottants, ...) et des ObjectProperty aux variables d'instance de type objet. Par exemple la classe suivante :

```
class Escalier{
   private Etage etageSuperieur;
   private Etage etageInferieur;
```

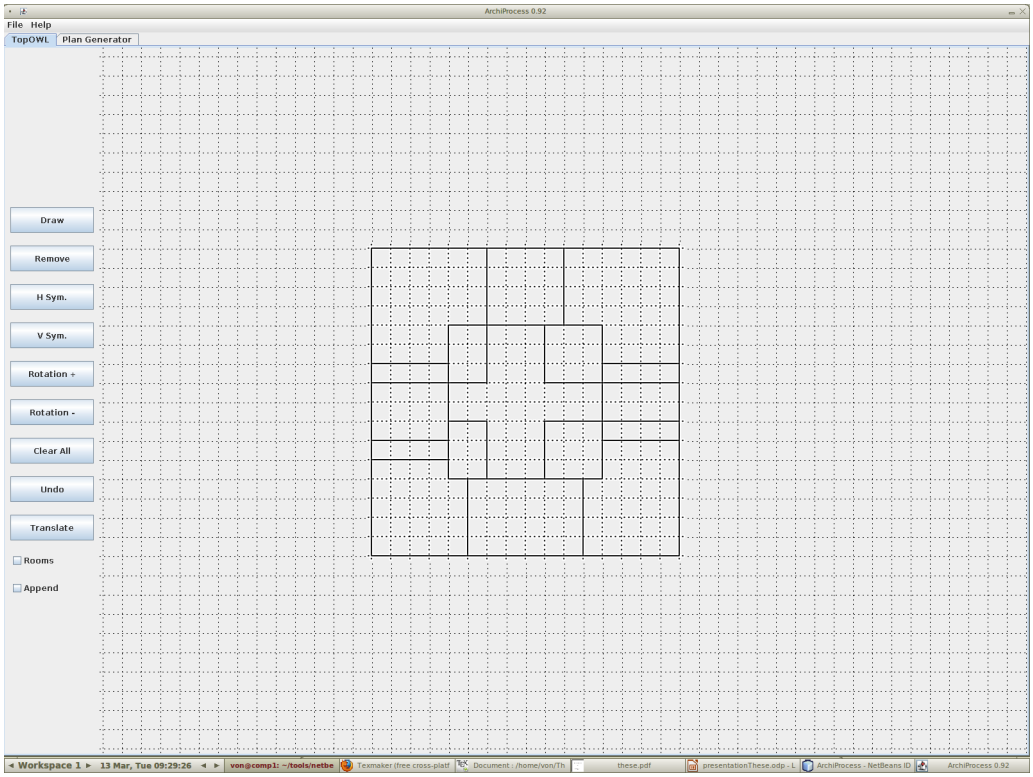


FIGURE 6.3 – Interface utilisateur permettant le dessin de schéma de composition

```
   private int   nombreMarche;
   private float hauteurMarche;
   public  Etage getEtageSuperieur() {return etageSuperieur};
   public  Etage getEtageInferieur() {return etageInferieur};
   public  int   getNombreMarche()   {return nombreMarche};
   public  float getHauteurMarche()  {return hauteurMarche};
   public  float getHauteurEscalier(){return nombreMarche*hauteurMarche};
}
```

est représenté par le concept OWL :

$$Escalier \equiv \exists etageSuperieur.Escalier \sqcap \exists etageInferieur.Escalier \sqcap \exists nombreMarche.Int \sqcap \exists hauteurMarche.Float \sqcap \exists hauteurEscalier.Float \quad (6.1)$$

La conversion se fait en listant par réflexion les méthodes de la classe Escalier et en générant les ObjetProperties et DataProperties correspondants. Le contenu des méthodes décrivant des contraintes entre les paramètres est cependant difficilement stockable en OWL. La conversion en sens inverse, de

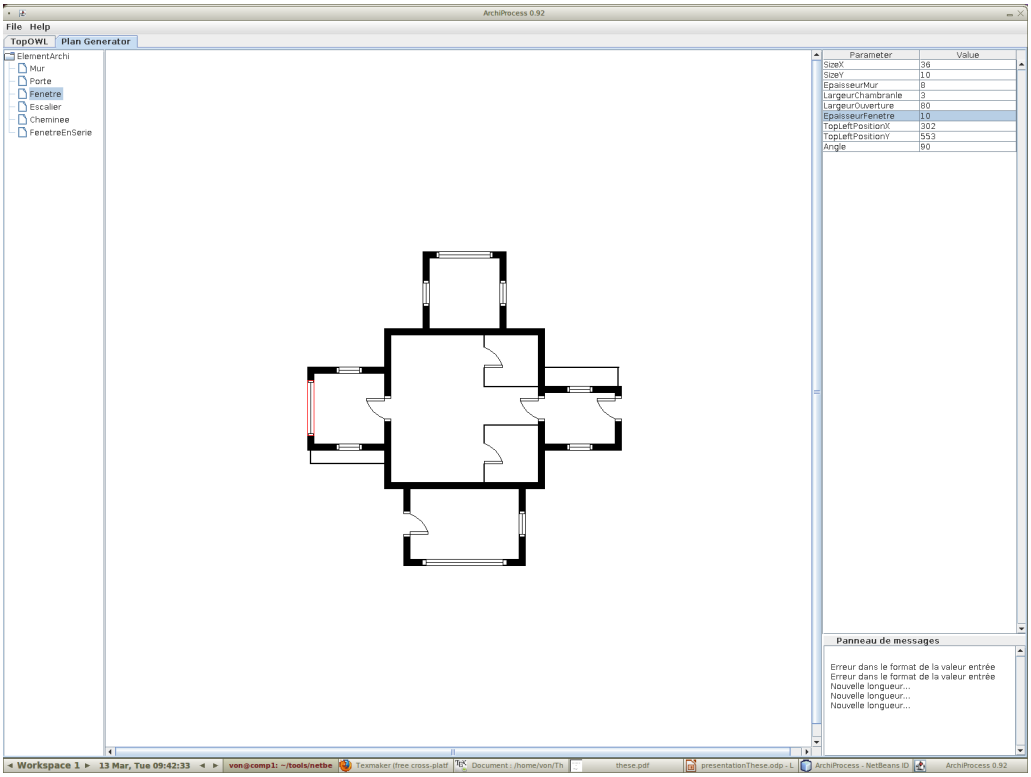


FIGURE 6.4 – Interface utilisateur permettant le positionnement et le paramétrage des éléments d'architecture

OWL vers JAVA, est faite en générant dynamiquement des objets à partir de la représentation OWL des éléments d'architecture.

6.5.2 Graphes en OWL et GraphML

Les connexions des éléments d'architecture entre eux et la connexions des pièces entre elles peuvent être facilement représentées sous la forme de graphes. En OWL, il est possible de représenter ces graphes sous forme de ABox. Cependant, l'utilisation d'algorithmes sur ces graphes nécessite la présence d'une librairie externe. La librairie JAVA la plus utilisée est JGraphT[3]. Elle permet l'exportation et l'importation des graphes dans le format GraphML[18]. Ce format permet la gestion de graphes annotés. Il est notamment utilisé par l'éditeur de graphes yEd[4] qui permet d'exporter des dessins géométriques en GraphML.

6.5.3 Utilisation de la réflexion

La réflexion permet de traiter une classe JAVA comme un objet. Il est ainsi possible pour une classe de type inconnu de déterminer son constructeur, ses variables publiques et l'ensemble de ses méthodes. Cette technique permet d'importer des éléments d'architecture définis dans le langage Processing sans avoir à modifier le logiciel de conception architecturale écrit en JAVA.

6.5.4 Utilisation de OWL API

OWL API est l'implémentation de référence pour la création, la manipulation et la sérialisation d'ontologies OWL. Il permet de générer des fichiers OWL directement utilisables par le logiciel Protégé et par les différents raisonneurs comme Pellet[89], HermiT[68], Racer[43]... L'ensemble des éléments d'architecture d'un projet ainsi que leurs différents paramètres (taille, position, forme...) sont représentés dans un fichier OWL. Le schéma de composition est lui aussi représenté dans un fichier OWL.

6.5.5 Utilisation de JGraphT

JGraphT[3] est une librairie permettant de traiter différents types de graphes comme les graphes dirigés ou non, avec ou sans poids, les multi-graphes, les pseudo-graphes... Cette librairie implémente aussi un ensemble d'algorithmes standard sur des graphes comme l'algorithme du plus court chemin, des circuits eulériens, des cycles hamiltoniens, la détection de cycles...

6.5.6 Utilisation de JRacer

JRacer est une API JAVA qui permet d'envoyer des requêtes à un serveur Racer. Il est ainsi possible d'accéder à toutes les fonctionnalités du raisonneur Racer notamment l'abduction[43].

Chapitre 7

Conclusion

Le but premier de cette thèse est de décrire un ensemble de techniques permettant l'intégration d'une base de connaissance spatiale dans un outil d'aide à la conception architecturale.

Pour cela nous avons choisi, avec l'aide des travaux de Pierre Pellegrino[75][76] et d'Emmanuelle Jeanneret[52], d'utiliser une épistémologie de l'architecture basée sur des trames, des schémas de composition et des éléments d'architecture. Nous avons ainsi pu constituer un modèle permettant de représenter l'évolution d'un projet architectural, selon une méthodologie donnée, depuis l'esquisse la plus simple, un carré par exemple, jusqu'à un plan d'architecture.

Pour modéliser les éléments de conception architecturale, un formalisme spatio-terminologique a été développé en combinant différentes logiques déjà existantes. Ces logiques ont été choisies non seulement en fonction de leur expressivité mais aussi de leur décidabilité. C'est-à-dire de l'existence de procédure de raisonnement automatique. L'avantage d'utiliser des formalismes standard est de pouvoir utiliser les moteurs de raisonnement déjà existants comme notamment le moteur *Racer* qui permet d'effectuer des raisonnements abductifs.

Une base de connaissance a été utilisée pour stocker les différents éléments de conception, comme les trames ou les schémas de composition, ainsi que pour stocker une théorie générale de l'architecture. Cette base de connaissance permet au concepteur d'utiliser à la fois des éléments de style, issus d'une analyse théorique de la pratique architecturale, et des éléments de conception utilisés dans des projets antérieurs.

En appliquant les procédures de raisonnement automatiques, disponibles dans les formalismes choisis, à la base de connaissance constituée, il devient possible de classifier les éléments déjà créés, de vérifier la consistance d'ensembles de contraintes, d'appliquer des règles, ainsi que de créer de nouvelles instances par abduction.

Un prototype démontrant la faisabilité d'un tel système a été développé. Ce prototype permet de dessiner des schémas de composition par application d'opérations géométriques. Ces schémas sont représentés en utilisant les formalismes du chapitre 5, ils peuvent être exportés dans le format standard de la logique descriptive (OWL) afin d'utiliser les différents raisonneurs existants. Les éléments d'architecture développés avec le langage *Processing* peuvent être importés, placés et composés sur le schémas de composition puis exportés en OWL.

Nous avons ainsi pu décrire et mettre en œuvre des techniques permettant l'intégration des outils d'aide à la conception architecturale avec les bases de connaissance. Différentes tâches de raisonnement, comme la vérification de la consistance d'un projet par rapport à un ensemble de règles ou la classification d'éléments de projet dans une base de connaissance contenant des projets antérieurs, ont été rendues possibles par la modélisation logique des éléments de conception architecturale.

Dans cette thèse, nous avons développé un ensemble d'éléments logiques et informatiques nécessaires à la construction d'un langage dédié à l'architecture (*Domain Specific Language*) qui combinent une approche terminologique et spatiale et qui permettent à un utilisateur d'exprimer ses problématiques architecturales. Ces éléments permettent d'expérimenter des modes d'interconnections entre les outils d'aide à la conception architecturale et les bases de connaissance, mais ne forment cependant pas encore, pour l'instant, un ensemble homogène.

D'autres domaines que celui de l'architecture peuvent être modélisés par le biais des mêmes formalismes. Par exemple, tout ce qui touche au domaine juridique et réglementaire. Des bases de connaissance spécifiques peuvent être constituées et connectées au système de projets. Des contraintes juridiques comme les volumes maximum autorisés, les distances sur les parcelles ou les distances de sécurité peuvent ainsi être modélisées.

Plusieurs axes de recherche restent à explorer. Le développement d'un langage interprété permettant d'exprimer de manière fluide (*Literate Programming*) des problématiques architecturales reste encore à effectuer. L'introduction d'une dimension diachronique, dans les représentations logiques, permettrait d'analyser l'évolution des projets architecturaux comme un processus en soi et serait certainement pertinente du point de vue de la théorie de l'architecture. L'introduction d'une dimension statistique dans la représentation logique permettrait quant à elle de traiter, avec par exemple des réseaux de neurones, des ensembles de plans architecturaux pour en tirer des concepts et des règles logiques propres à définir des styles nouveaux à partir de styles existants. Les projets en cours de conception se trouvent fréquemment dans des états inconsistants. Il existe des algorithmes permettant de déterminer

les axiomes entrainant l'inconsistance. Il n'existe cependant pas encore de mécanismes d'inférence permettant de traiter des base de connaissance inconsistantes.

Bibliographie

[1] Minsky's frame system theory. In *TINLAP '75 : Proceedings of the 1975 workshop on Theoretical issues in natural language processing*, pages 104–116, Morristown, NJ, USA, 1975. Association for Computational Linguistics.

[2] Gecode : An Open Constraint Solving Library,Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP), Paris, France, May,, 2008.

[3] *JGraphT – a free Java graph library*, 2009.

[4] *yEd – a graph editor*, 2011.

[5] Marco Aiello, Ian E. Pratt-Hartmann, and Johan F.A.K. van Benthem. *Handbook of Spatial Logics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[6] Leon Battista Alberti and Cecil Grayson. *On painting and On sculpture. The Latin texts of De pictura and De statua [by] Leon Battista Alberti. Edited with translations, introduction and notes by Cecil Grayson.* Phaidon [London], 1972.

[7] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11) :832–843, 1983.

[8] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. The MIT Press, 2008.

[9] F. Baader, D. Calvanese, G. De Giacomo, P. Fillottrani, E. Franconi, B. Cuenca Grau, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, R. Moeller, B. Parsia, P. Patel-Schneider, R. Rosati, B. Suntisrivaraporn, and S. Tessaris. Formalisms for Representing Ontologies : State of the Art Survey, May 2005.

[10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[11] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th international joint conference on Artificial intelligence - Volume 1*, pages 452–457, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[12] Jie Bao, Doina Caragea, and Vasant Honavar. A Tableau-Based Federated Reasoning Algorithm for Modular Ontologies. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '06, pages 404–410, Washington, DC, USA, 2006. IEEE Computer Society.

[13] Tim Berners-Lee. Semantic Web Road Map, 1998.

[14] Mehul Bhatt, Joana Hois, and Oliver Kutz. Modelling Form and Function in Architectural Design. *Submitted draft*, 2010.

[15] Meghyn Bienvenu. Complexity of Abduction in the EL Family of Lightweight Description Logics. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR08)*. AAAI Press, 2008. To appear.

[16] Paul V. Biron and Ashok Malhotra, editors. *XML Schema Part 2 : Datatypes*. W3C Recommendation. W3C, second edition, October 2004.

[17] Pim Borst, Hans Akkermans, and Jan Top. Engineering Ontologies. *International Journal of Human-Computer Studies*, 46(2-3) :365–406, 1997.

[18] U Brandes, M Eiglsperger, I Herman, M Himsolt, and MS Marshall. GraphML Progress Report, Structural Layer Proposal. In P Mutzel, M Junger, and S Leipert, editors, *Graph Drawing - 9th International Symposium, GD 2001 Vienna Austria,*, pages 501–512, Heidelberg, 2001. Springer Verlag.

[19] Anthony G. Cohn and Shyamanta M. Hazarika. Qualitative Spatial Representation and Reasoning : An Overview. *Fundam. Inform.*, 46(1-2) :1–29, 2001.

[20] Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, Marina Mongiello, and Francesco M. Donini. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. In *ICEC '04 : Proceedings of the 6th international conference on Electronic commerce*, pages 41–50, New York, NY, USA, 2004. ACM.

[21] François de Bertrand de Beuvron and Amadou Coulibaly. Applying Description Logic to Product Behavioral Design within Advanced CAD

Systems. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *Description Logics*, volume 81 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[22] Tommaso Di Noia, Eugenio Di Sciascio, and Francesco M. Donini. Semantic Matchmaking as Non-Monotonic Reasoning : A Description Logic Approach. *Journal of Artificial Intelligence Research (JAIR)*, 29 :269–307, 2007.

[23] Nick Drummond, Alan L. Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai Wang, and Julian Seidenberg. Putting OWL in Order : Patterns for Sequences in OWL. In *OWLED*, 2006.

[24] F. Dylla, L. Frommberger, J. O. Wallgrun, and D. Wolter. SparQ : A toolbox for qualitative spatial representation and reasoning. In *Workshop at KI'06*, page 79, 2006.

[25] E. Nanchen. *Distances entre classes d'isomorphisme de graphes*. PhD thesis, Université de Genève, 1998.

[26] E. Nanchen et D. Coray avec S. Cirilli et E. P. Jeanneret sous la direction de D. Coray et P. Pellegrino. GED, Graph Editor. CRAAL, Centre de Recherche en Architecture et Architecturologie, 1996.

[27] Charles Eastman and Max Henrion. Glide : a language for design information systems. *SIGGRAPH Comput. Graph.*, 11 :24–33, July 1977.

[28] O. Stock (Ed.). *Spatial and Temporal Reasoning*. Springer, 1997.

[29] Corinna Elsenbroich, Oliver Kutz, and Ulrike Sattler. A case for abductive reasoning over ontologies. In *in Proc. OWL : Experiences and Directions*, pages 10–11, 2006.

[30] Bruno Errico and Gabriele Novembri. KAAD : A Support System for Architectural Design. In *ICTAI*, pages 792–795, 1994.

[31] Ernest Friedman-Hill. *Jess in Action : Java Rule-based Systems*. Manning, Greenwich, CT, 2003.

[32] Thomas Froese, Francois Grobler, John Ritzenthaler, Kevin Yu, and Sheryl Staub. Industry Foundation Classes for Project Management - A Trial Implementation, 1999.

[33] M. Genesereth. Knowledge Interchange Format. Proposed draft, 1998.

[34] John S. Gero. An Overview of knowledge engineering and its relevance to CAAD. In *CAAD Futures*, pages 107–119, 1986.

[35] John S. Gero. *A locus for knowledge-based systems in CAAD education*, pages 49–60. MIT Press, Cambridge, MA, USA, 1990.

[36] John S. Gero. AI EDAM at 20 : Artificial intelligence in designing. *Artif. Intell. Eng. Des. Anal. Manuf.*, 21 :17–18, January 2007.

[37] Asuncion Gomez-Perez, Mariano Fernandez-Lopez, and Oscar. Corcho. *Ontological Engineering*. Springer Verlag, London, 2004.

[38] Roop K. Goyal and Max J. Egenhofer. Similarity of Cardinal Directions. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, SSTD '01, pages 36–58, London, UK, UK, 2001. Springer-Verlag.

[39] E. Grädel and M. Otto. On Logics with Two Variables. *Theoretical Computer Science*, 224 :73–113, 1999.

[40] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs : combining logic programs with description logic. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 48–57, New York, NY, USA, 2003. ACM.

[41] T. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2) :199–220, 1993.

[42] Nicola Guarino. Some Ontological Principles for Designing Upper Level Lexical Resources. *CoRR*, cmp-lg/9809002, 1998.

[43] V. Haarslev and R. Möller. Racer : A Core Inference Engine for the Semantic Web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20*, pages 27–36, 2003.

[44] V. Haarslev, R. Möller, and M. Wessel. Visual Spatial Query Languages : A Semantics Using Description Logic. In P. Olivier, M. Anderson, and B. Meyer, editors, *Diagrammatic Representation and Reasoning*. Springer-Verlag, 2001.

[45] J. Hendler and D. L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6) :67–73, 2000.

[46] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language : Primer*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-primer/`.

[47] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 57–67. AAAI Press, 2006.

[48] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL : A Semantic Web Rule Language

Combining OWL and RuleML. W3c member submission, World Wide Web Consortium, 2004.

[49] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical Reasoning for Expressive Description Logics. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *LPAR*, volume 1705 of *Lecture Notes in Computer Science*, pages 161–180. Springer, 1999.

[50] Bo Hu, Yannis Kalfoglou, Harith Alani, Paul Lewis, and Nigel Shadbolt. N.shadbolt. semantic metrics. In *In Proceedings of the 15 th International Conference on Knowledge Engineering and Knowledge Management (EKAW'06), Podebrady, Czech*, 2006.

[51] James G. Schmolze and Bolt Beranek and Newman Inc. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9 :171–216, 1985.

[52] Emmanuelle P. Jeanneret. *Langage et contexte, Géographie de la maison et architecture des territoires*. Thèse de doctorat, Université Polytechnique de Catalogne, 2003.

[53] Yehuda E. Kalay and William J. Mitchell. *Architecture's New Media : Principles, Theories, and Methods of Computer-Aided Design*. The MIT Press, 2004.

[54] Szymon Klarman. ABox Abduction in Description Logic. Master's thesis, Universiteit van Amsterdam, June 2008.

[55] Joseph B. Kopena and William C. Regli. Design Repositories On The Semantic Web With Description-Logic Enabled Services. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universitat, Berlin, Germany, September 7-8*, 2003.

[56] Petr Kremen and Evren Sirin. SPARQL-DL Implementation Experience. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of the Fourth OWLED Workshop on OWL : Experiences and Directions Washington, DC (metro), 1-2 April 2008*, volume 496 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[57] O. Kutz, H. Sturm, N.-Y. Suzuki, F. Wolter, and M. Zakharyaschev. Logics of Metric Spaces. *ACM Transactions on Computational Logic (TOCL)*, 4(2) :260–294, 2003.

[58] O. Kutz, F. Wolter, and M. Zakharyaschev. A Note on Concepts and Distances. In *Proceedings of the International Workshop on Description Logics (DL-2001)*, Stanford, 2001.

[59] O. Kutz, F. Wolter, and M. Zakharyaschev. Connecting abstract description systems. In *Proceedings of the 8th Conference on Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 215–226. Morgan Kaufmann, 2002.

[60] O. Lassila and Deborah L. McGuinness. The Role of Frame-Based Representation on the Semantic Web. Technical Report KSL-01-02, Stanford University, Stanford, 2001.

[61] A. Loos. *Ornament and Crime*. Innsbruck, Reprint Vienna, 1930.

[62] Carsten Lutz and Maja Milicic. A Tableau Algorithm for Description Logics with Concrete Domains and General TBoxes. *J. Autom. Reasoning*, 38(1-3) :227–259, 2007.

[63] Robert M. MacGregor. Inside the LOOM description classifier. *SIGART Bull.*, 2 :88–92, June 1991.

[64] Deborah L. McGuinness. The CLASSIC Knowledge Representation System : Implementation, Applications, and Beyond. In *Description Logics*, pages 80–86, 1991.

[65] Deborah L. McGuinness, Richard Fikes, James A. Hendler, and Lynn Andrea Stein. DAML+OIL : An Ontology Language for the Semantic Web. *IEEE Intelligent Systems*, 17(5) :72–80, 2002.

[66] William J. Mitchell. *The Logic of Architecture : Design, Computation, and Cognition*. MIT Press, Cambridge, MA, USA, 1st edition, 1990.

[67] Yoshihiro Mizoguchi and Yasuo Kawahara. Relational Graph Rewritings. *Theor. Comput. Sci.*, 141(1&2) :311–328, 1995.

[68] Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. Structured Objects in OWL : Representation and Reasoning. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proc. of the 17th Int. World Wide Web Conference (WWW 2008)*, pages 555–564, Beijing, China, April 21–25 2008. ACM Press.

[69] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. In *Journal of Web Semantics*, pages 549–563. Springer, 2004.

[70] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, pages 2–9, New York, NY, USA, 2001. ACM.

[71] Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello. Abductive Matchmaking using Description Logics. In *IJCAI*, pages 337–342, 2003.

[72] Jianjun Oung, Meera Sitharam, Brandon Moro, and Adam Arbree. FRONTIER : fully enabling geometric constraints for feature-based modeling and assembly.

[73] W3C OWL Working Group. *OWL 2 Web Ontology Language : Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[74] Sebastian Rudolph Pascal Hitzler, Markus Krötzsch. OWL 2 Rules (Part 2). Tutorial at ESWC2009, May 31 2009.

[75] Pierre Pellegrino. *Le Sens de l'Espace, Les Grammaires et les Figures de l'Etendue, Livre III*. La bibliothèque des Formes, Ed. Economica, 2003.

[76] Pierre Pellegrino. *Le Sens de l'Espace, Le Projet architectural, Livre IV*. La bibliothèque des Formes, Ed. Economica, 2007.

[77] Pierre Pellegrino, Daniel Coray, Gilles Falquet, Emmanuelle P. Jeanneret, and Mathieu Vonlanthen. Modélisation du projet et formalisation du savoir architectural, L'élaboration du projet architectural dans un espace virtuel. Rapport intermédiaire No 1, Université de Genève, 31 octobre 2008.

[78] Vitruvius Pollio. and Claude Perrault. *Les dix livres d'architecture de Vitruve, corrigez et tradvits nouvellement en Francois, avec des notes et des figures* . J. B. Coignard, Paris, 1673.

[79] Sviataslau Pranovich, Jarke J. Van Wijk, and Huub Van De Wetering. Abstract An Architectural Design System for the Early Stages, 2008.

[80] M. Ross Quillian. *Semantic Information Processing*, chapter Semantic memory. MIT Press, Cambridge, MA, 1968.

[81] David A. Randell, Zhan Cui, and Anthony Cohn. A Spatial Logic Based on Regions and Connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning : Proceedings of the Third International Conference*, pages 165–176. Morgan Kaufmann, San Mateo, California, 1992.

[82] Casey Reas, Ben Fry, and John Maeda. *Processing : A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2007.

[83] Jochen Renz. Maximal tractable fragments of the region connection calculus : A complete analysis. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99*, pages 448–454. Morgan Kaufmann, 1999.

[84] Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1) :41–60, 2005.

[85] S. Cirilli en collaboration avec E. P. Jeanneret et E. Nanchen sous la direction de P. Pellegrino et D. Coray. TOP, Taxis Oriented Project. CRAAL, Centre de Recherche en Architecture et Architecturologie, 1996.

[86] Guus Schreiber and Mike Dean. OWL web ontology language reference. W3C recommendation, W3C, February 2004.

[87] H.A. Simon. *The science of the artificial*. MIT Press, Camridge, 1969.

[88] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet : A practical OWL-DL reasoner. *Web Semantics : Science, Services and Agents on the World Wide Web*, 5(2) :51–53, June 2007.

[89] E Sirin, B Parsia, BC Grau, A Kalyanpur, and Y Katz. Pellet : A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2) :51–53, June 2007.

[90] Evren Sirin and Bijan Parsia. SPARQL-DL : SPARQL Query for OWL-DL. In *OWLED 2007 : Proceedings of the Third International Workshop on OWL : Experiences and Directions, Innsbruck, Austria*, 2007.

[91] George Stiny. *Shape. Talking about seeing and doing.* MIT Press, Cambridge, Ma., 2006.

[92] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors. *Modular Ontologies : Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*. Springer, 2009.

[93] L. Sullivan. The tall office building artistically considered. *Lippincott's Magazine*, 1896.

[94] Ivan E. Sutherland. Sketchpad : A Man-Machine Graphical Communication System. In E. Calvin Johnson, editor, *Proceedings of the 1963 Spring Joint Computer Conference*, volume 23, pages 329–346. American Federation of Information Processing Societies, Spartan Books, 1963.

[95] P. Szalapaj. *CAD Principles For Architectural Design : Analytical Approaches to Computational Representation of Architectural Form.* Architectural Press, Oxford, 2001.

[96] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L. McGuinness. Integrity Constraints in OWL. In *AAAI*, 2010.

[97] K. Terzidis. *Algorithmic architecture.* Architectural Press, 2006.

[98] The OSGi Alliance. OSGi service platform core specification, release 4.3. `http://www.osgi.org/Specifications`, 2011.

[99] Yuri A. Tijerino and Riichiro Mizoguchi. MULTIS II : Enabling End-Users to Design Problem-Solving Engines via Two-Level Task Ontologies. In *EKAW*, pages 340–359, 1993.

[100] Mike Uschold, Mike Uschold, Michael Grüninger, and Michael Gruninger. Ontologies : Principles, Methods and Applications. *Knowledge Engineering Review*, 11 :93–136, 1996.

[101] Gertjan van Heijst, A. Th. Schreiber, and Bob J. Wielinga. Using explicit ontologies in KBS development. *Int. J. Hum.-Comput. Stud.*, 46(2) :183–292, 1997.

[102] Jan Wallgrün, Lutz Frommberger, Diedrich Wolter, Frank Dylla, and Christian Freksa. Qualitative Spatial Representation and Reasoning in the SparQ-Toolbox. In Thomas Barkowsky, Markus Knauff, Gérard Ligozat, and Daniel R. Montello, editors, *Spatial Cognition V Reasoning, Action, Interaction* , volume 4387, chapter 3, pages 39–58. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[103] Filip Zelezný and Nada Lavrac, editors. *Inductive Logic Programming, 18th International Conference, ILP 2008, Prague, Czech Republic, September 10-12, 2008, Proceedings*, volume 5194 of *Lecture Notes in Computer Science*. Springer, 2008.

[104] Yingzhong Zhang and Xiaofang Luo. Description logic representation for semantic concepts of feature. In *11th IEEE International Conference on Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09.* , pages 544 –547, aug. 2009.