Introduction de raisonnement dans un outil industriel de gestion des connaissances

Olivier Carloni

Académie de Montpellier
Université Montpellier II
— **Sciences et Techniques du Languedoc** —

Thèse

présentée au Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier pour obtenir le diplôme de doctorat

Spécialité	:	**Informatique**
Formation doctorale	:	**Informatique**
École doctorale	:	**Information, Structures, Systèmes**

Introduction de raisonnement dans un outil industriel de gestion des connaissances

par

Olivier Carloni

Soutenue le 24 novembre 2008 devant le Jury composé de :

Jean Charlet, Professeur, INSERM/Ecole Centrale Paris ..... rapporteur
Jean Delahousse, PDG, Société Mondeca ..... examinateur
Ollivier Haemmerlé, Professeur, Université Toulouse Le Mirail ..... rapporteur
Danièle Hérin, Professeur, Université Montpellier II ..... examinatrice
Michel Leclère, Maître de Conférences, Université Montpellier II . co-directeur de thèse
Marie-Laure Mugnier, Professeur, Université Montpellier II ..... co-directrice de thèse

Table des matières

Remerciements

Avant de présenter le travail que j'ai réalisé durant ces dernières années, je souhaiterais adresser une série de remerciements à ceux que j'ai connus et appréciés tout au long de mon parcours. J'adresse mes premiers et plus forts remerciements à Marie-Laure Mugnier et surtout à Michel Leclère pour leur engagement dans la supervision de ma thèse, la qualité de leurs conseils et de leur soutien. Je remercie également très chaleureusement le président directeur général de la société Mondeca Jean Delahousse pour son écoute attentive, ses qualités humaines et son suivi en tant que tuteur industriel. J'embrasse avec beaucoup d'affection mes parents, ma soeur, son ami Julien et ma douce Marina pour leur patience et leur aide dans les derniers moments les plus intenses de la thèse (surtout toi Marina qui était à mes cotés alors que j'avais le caractère difficile). Bien évidement je ne vous ai pas oublié mes amis de toujours et en particulier Jérome, Julien, Camille, Laure, les deux Nicolas (le dentiste et le thésard), Anne-Lise, Aude et Pierre. J'ai également une pensée toute particulière pour Sylvain (alias sissou) que j'ai connu depuis ma tendre enfance et qui est et restera toujours un grand ami (si ce n'est dire LE). J'adresse aussi une bonne poignée de main bien virile à mon ami Jean Baylon avec qui j'ai eu d'interminables et passionnantes discussions philosophiques, philologiques et mathématiques... Mes derniers remerciements mais non les moindres s'adressent tout naturellement aux employés de la société Mondeca et aux membres de l'équipe RCR ; et à tous ceux que je n'ai pas cités.

Chapitre 1

Introduction

Le travail de thèse présenté dans ce document porte sur la conception d'un service de validation et d'enrichissement d'annotations pour un outil industriel de gestion des connaissances basé sur le langage des Topic Maps (TM). Un tel service nécessitant la mise en œuvre de raisonnements sur les connaissances, il a été nécessaire de doter le langage des TM d'une sémantique formelle. Ceci a été réalisé par l'intermédiaire d'une transformation réversible des TM vers le formalisme logique des graphes conceptuels qui dispose d'une représentation graphique des connaissances (les TM pouvant facilement en être munie d'une). La solution a été mise en oeuvre dans deux applications, l'une conçue pour la veille médiatique et l'autre pour la promotion de ressources touristiques. Schématiquement, des annotations sont extraites automatiquement des documents selon le domaine concerné (actualité/économie ou tourisme) puis ajoutées à la base de connaissances. Elles sont ensuite fournies au service d'enrichissement et de validation qui les complète de nouvelles connaissances et décide de leur validité.

Les sections suivantes présentent les contextes scientifique et industriel au sein desquels ce travail s'inscrit et un résumé détaillé des résultats de la thèse.

Contexte scientifique

Le Web (World Wide Web, voir [BLCG92]) est un cadre d'échange public d'informations fondé sur le réseau Internet apparu à la fin des années 80. Il permet l'organisation sous forme de sites web de ressources interactives dont le contenu codé sur support électronique est essentiellement destiné à une exploitation humaine. Les informations circulant sur le Web sont en partie structurées par le format standard hypertexte (HTML) proposé en 1989 par Tim Berners-Lee [BL88]. Ce format permet entre autres d'établir des relations entre documents numériques et d'en présenter le contenu pour une lecture humaine. La puissance et liberté d'expression que confère le Web à l'individu est telle qu'au cours des deux décennies suivant sa naissance la taille du Web ne va cesser de

croître pour atteindre un nombre dépassant actuellement les 100 millions de sites.

Cette surabondance d'informations conduit à l'émergence de nouveaux besoins, certains d'ordre quantitatif tel que celui de maintenir à jour efficacement d'importants volumes de données ; d'autres d'ordre plutôt qualitatif comme la création, la recherche ou l'exploitation de l'information ou du service le plus pertinent par rapport à certaines exigences. En 2001, T. Berners-Lee, J. Hendler et O. Lassila proposent dans [BLHL01] une nouvelle infrastructure basée sur celle du Web répondant en théorie au besoin d'une structuration plus pertinente de l'information : le Web Sémantique.

Très brièvement : le Web Sémantique prévoit de faciliter la recherche d'informations et d'accroître l'interopérabilité des agents (qu'ils soient humains ou logiciels) en formalisant la connaissance que recèle le contenu non structuré du Web ; c'est à dire les informations créées et comprises par l'homme mais pas encore exploitables par les machines. Le Web Sémantique fonde la description du contenu des ressources sur des langages intelligibles autant par l'homme que par la machine et offrant une meilleure expressivité[1] que le format hypertexte. L'accès à cette connaissance et le perfectionnement des traitements dont elle peut faire l'objet doit conduire à améliorer la recherche d'information en allégeant les opérations parfois laborieuses de classement, de recoupement et de sélection des résultats.

Bien que le concept de Web Sémantique laisse espérer de considérables progrès en matière de traitement de l'information, il n'en reste pas moins défini dans la continuité du concept de Web ; ce dernier permettant déjà dans une moindre mesure de répondre à des besoins complexes.

Citons par exemple les systèmes de gestion de contenu (SGC) (cf. [Boi01b] et [Boi01a]) qui sont apparus au milieu des années 90. Ces systèmes fonctionnent à partir du format hypertexte et offrent des services relativement perfectionnés. Ils ont été conçus pour mettre à jour dynamiquement d'importantes quantités d'informations tout en offrant des possibilités de travail collaboratif, de mise en ligne de contenu, de structuration et de séparation claire entre gestion du contenu et gestion de la forme qu'il prend à la publication. Les SGC offrent des méthodes variées pour exploiter ce contenu. Elles sont basées sur l'ajout de références (par liens hypertexte), sur des méthodes de recherche "plein texte" et des procédures de tris. Ces méthodes sont parfois complétées par des mécanismes de catégorisation (sous forme de taxonomies par exemple) et d'indexation de contenu à partir des termes d'un thesaurus. Au fil du temps, certains SGC [SBA+02] se sont perfectionnés jusqu'à proposer une dimension "sémantique" à la gestion de contenu ; brouillant ainsi la frontière entre la gestion de contenu *pure* et la gestion des connaissances.

Le Web des années 2000 permet aussi l'intégration et l'automatisation d'agents exploitant les fonctions du réseau même s'il est vrai, comme le souligne [KT05], que le Web

[1] C'est à dire que ces langages permettent de décrire des nuances inexpressibles dans le format hypertexte.

sémantique laisse espérer une véritable "orchestration automatisée" de ces agents. Les langages à fort pouvoir d'expression devraient servir à décrire la nature des services offerts par ces agents : c'est la filière des services web sémantiques (dont WSDL [CCMW01] est un composant élémentaire).

Dans une certaine mesure, à ses débuts, le Web permet déjà d'organiser le contenu d'une ressource sous une forme exploitable par la machine. Le format hypertexte offre la possibilité de décrire une ressource web sous une forme structurée bien qu'elle soit relativement pauvre au regard de la complexité du contenu à décrire. Ces descriptions sont associées à la ressource et demeurent accessibles via le Web. Elles sont qualifiées d'*annotation* ou de *métadonnée*, bien qu'à l'heure actuelle les distinctions entre ces deux termes demeurent floues. Du point de vue de l'usage, l'annotation est une remarque, une note explicative et la métadonnée est une valeur codée dans un certain format. La première exprime une information sur un document et la seconde une donnée. Ces descriptions peuvent être séparées en deux classes, celles qualifiées d'*objectives* qui concernent le support électronique de la ressource (par exemple la date de création/modification ou l'auteur d'un fichier) et celles dites *subjectives* qui portent sur son contenu et dépendent de son interprétation (par exemple, les mots clés qui décrivent le contenu). D'un point de vue fondamental, il ne semble pas y avoir de différence entre la notion de métadonnée et celle d'annotation. Comme le souligne [PG05], certains s'accordent à dire que l'annotation d'un document devient une métadonnée lorsqu'elle en est séparée pour être entreposée dans une base et exploitée indépendamment de la ressource ; ou encore, qu'une métadonnée accompagne la ressource et respecte un format précis de codage alors qu'une annotation a été ajoutée au contenu de la ressource au cours d'un processus et peut être rédigée en langage naturel ; ce qui conduit à la rapprocher d'une description subjective puisqu'elle dépend d'une interprétation (qui peut être humaine ou logicielle).

Quoi qu'il en soit, les anciens formats utilisés pour définir des métadonnées ou annotations ne sont pas suffisants pour décrire avec précision le contenu d'une ressource. Le Web Sémantique prévoit des annotations et métadonnées qualifiées de "sémantiques" en le sens qu'elles expriment des informations complexes exploitées par des mécanismes sophistiqués pour fournir des services "intelligents" à l'utilisateur. Ces mécanismes s'appuient parfois sur les méthodes développées dans le domaine de l'intelligence artificielle, et plus précisément le domaine de la *représentation des connaissances et raisonnements* (abrégé en *RCR*).

D'un point de vue général, un algorithme réalise des manipulations de symboles pour calculer une expression *résultat* en fonction d'une expression *donnée*. En *RCR* le statut accordé à ces expressions s'éloigne de celui de *donnée* et évoque plutôt celui de *connaissance* et les manipulations de symboles correspondent à des inférences ou raisonnements. Comme le signale [Cha02], il est difficile d'établir une distinction formelle entre les notions d'information, de donnée et de connaissance. La signification des trois termes se chevauche ; le sens de l'un se situant dans la continuité de celui des deux autres. Il en va

de même pour les notions d'algorithme (au sens général) et de raisonnement automatique qui héritent de cette ambiguïté.

[Baa96] s'intéresse davantage aux langages de *RCR* munis d'une interprétation logique et, en introduction, aux distinctions qui peuvent être établies entre la programmation classique et les méthodes utilisées en représentation des connaissances. En algorithmique, un besoin est formalisé sous la forme d'un problème P identifié par un nom, une donnée D et la spécification formelle d'un résultat R énoncée pour D. On considère, par exemple, le problème qui consiste à savoir si deux sommets d'un graphe sont connectés l'un à l'autre. Dans cet exemple, la donnée est composée des deux sommets et du graphe. Le résultat est *vrai* si les deux sommets donnés sont "connectés" dans le graphe ; sachant que deux sommets sont "connectés" dans un graphe s'il existe dans ce graphe un chemin de l'un à l'autre. Une fois le problème défini, on cherche le moyen d'obtenir le résultat R en fonction de D généralement sous la forme d'un algorithme A qui retourne le résultat en un nombre fini d'étapes. D'une certaine manière, l'algorithme A code de manière *procédurale* les contraintes C que doivent respecter deux expressions D et R pour être admises comme donnée et résultat *du problème* P en particulier. Dans notre example, cela revient à écrire dans un langage de programmation un programme de parcours de graphe qui associerait le résultat *vrai* si pour un graphe donné le couple de sommets spécifié en entrée respecte la définition de "connectés". Un tel algorithme utilise des structures de données et une procédure de résolution *spécifiques* au problème énoncé. Dans cette conception, la donnée, le résultat et l'algorithme sont assez clairement dissociées. [Baa96] soulève qu'en *RCR*, la donnée D, le résultat R et les contraintes C correspondent à des objets qui appartiennent à une même classe : la *connaissance*. On considère qu'on peut coder de manière *déclarative* les expressions D, R et les contraintes C en trois expressions d, r et c d'un *même* langage $\mathcal{L}$ de *RCR*, afin de déterminer - en utilisant des mécanismes intrinsèques au langage $\mathcal{L}$ (dits mécanismes de *raisonnement*) - si D et R vérifient les critères appropriés pour être considérés comme *donnée* et *résultat* du problème P. Dans notre example, la résolution du problème P ne repose plus sur l'algorithme précédent mais sur un codage dans un langage déclaratif du graphe et de la notion "connecté" (voir fig. 1.1). Pour savoir si deux sommets s_x et s_y sont effectivement connectés, il suffit d'interroger le système en lui fournissant l'assertion à vérifier, ici : *connecté(x,y)*. Dans cette vision, les mécanismes de résolution ne sont plus spécifiques au problème P puisqu'ils ont été définis de façon générique pour toute expression du langage $\mathcal{L}$. La distinction proposée par [Baa96] tient donc en ce que contrairement au point de vue algorithmique, en *RCR* les expressions d, r et c sont d'une part formulées déclarativement dans un même langage L lui même intrinsèquement pourvu de *mécanismes génériques de raisonnement* ; d'autre part, d, r et c sont considérés comme les données de ces *mécanismes de raisonnement*.

Conjonction de *directement-connecté*(x, y) pour tout arc (s_x, s_y) du graphe.
Introduction de "connecté" : *directement-connecté*$(x, y) \rightarrow$ *connecté*(x, y)
Transitivité de "connecté" : *connecté*$(x, y) \wedge$ *connecté*$(y, z) \rightarrow$ *connecté*(x, z)

FIG. 1.1 – Le codage dans un langage déclaratif (ici, la logique du premier ordre sans fonction) d'un graphe en entrée et des contraintes à respecter pour résoudre le problème P.

Etant difficile d'aboutir à une distinction formelle entre les notions de donnée/algorithme et connaissance/raisonnement, il est raisonnable de proposer une distinction fondée sur l'usage : lorsqu'un symbole représentant une information est destiné à modifier le comportement d'un programme il constitue une donnée et lorsqu'il est voué à être humainement interprété dans un certain contexte il est perçu comme connaissance.

Les symboles représentant les connaissances du domaine modélisé sont pris en charge par des systèmes dits *à base de connaissance* (SBC) et s'organisent en des structures respectant le *langage de représentation des connaissances* associé au SBC. Ces langages sont parfois associés à une structure appelée "ontologie"[2] (voir [CBT05]) qui permet sous la forme d'un vocabulaire exploitable autant par l'homme que par la machine, de définir par des primitives, et des propriétés générales sur ces primitives, les catégories auxquelles appartiennent les objets du domaine modélisé. Les primitives et leurs propriétés sont prises en compte par les SBC au cours des raisonnements et conduisent à la construction de nouvelles représentations dans le même langage. Ces nouvelles représentations symboliques sont soumises à l'interprétation de l'utilisateur qui leur accorde un statut de connaissance et leur réserve un usage précis.

Comme le sens accordé à ces symboles reste subjectif, il peut conduire à des interprétations différentes voire contradictoires selon les individus. Pour éviter ces divergences, la sémantique de ces symboles est parfois établie suivant un consensus commun et fixée dans un langage formel : la *sémantique formelle*. Lorsque ce principe s'applique à une ontologie, on parle alors d'ontologie "formelle" (voir [Gua98]). Une sémantique formelle est en général exprimée dans un langage mathématique (la théorie des ensembles, par exemple) ou logique (la logique du premier ordre, par exemple). Ces langages sont sans doute les meilleurs candidats pour sa définition car ils détiennent un fort pouvoir d'expression, permettent d'établir des démonstrations rigoureuses et ont été utilisés et étudiés depuis plusieurs siècles par la communauté scientifique. L'existence d'une sémantique formelle pour un langage ne remet pas en question l'utilité de ce dernier : en effet, on pourrait se demander s'il n'est pas finalement plus simple d'utiliser directement

[2]Le terme *ontologie* prend ici un sens informatique et ne correspond pas à la définition qu'on lui accorde en philosophie (étude de l'être en tant qu'être). Il désigne donc ici une structure exploitable par un algorithme.

la sémantique formelle pour représenter les connaissances et abandonner le langage auquel cette sémantique formelle a été associée. En général, le langage choisi pour définir une sémantique formelle fournit souvent une expressivité élevée voire illimitée. Or les problèmes de calculabilité augmentent avec l'expressivité du langage. Par exemple, le problème de déterminer si entre deux formules de la logique du premier ordre l'une se déduit de l'autre est *semi-décidable* ; cela signifie que selon les cas on ne peut décider si oui ou non l'une des formules se déduit de l'autre. Néanmoins, lorsqu'on considère certains fragments moins expressifs de la logique du premier ordre, le problème de semi-décidabilité ne se pose plus (par exemple, le fragment existentiel positif conjonctif de la logique des prédicats ou le "guarded fragment"). Lorsque l'un de ces fragments décidables est choisi pour constituer la sémantique formelle d'un langage de *RCR*, ce dernier hérite de son pouvoir d'expression et de ses caractéristiques calculatoires mais est tenu, en plus, d'offrir une syntaxe et des opérations qui lui sont propres ; et c'est là où se situe son intérêt principal. En cas de doute sur le sens d'une expression établie dans la syntaxe, la sémantique formelle fait autorité pour éliminer l'ambiguïté ou la contradiction. En ce qui concerne la définition d'opérations directement associées au langage de *RCR*, elle constitue une option intéressante lorsque ce dernier s'inspire d'une théorie mathématique différente de celle de laquelle sa sémantique formelle est issue (par exemple, la théorie des graphes). Un tel choix peut être judicieux s'il est plus facile de définir des algorithmes efficaces dans cette théorie, par exemple. On peut ensuite vérifier si le résultat obtenu par application de l'opération sur les expressions du langage de *RCR* est cohérent en le comparant à celui défini par la sémantique formelle de la dite opération. D'un point de vue pratique, la sémantique formelle est principalement utilisée par les concepteurs ou spécialistes de SBC pour comparer les opérations symboliques réalisées et les différents langages de *RCR*. Il s'agit plus précisément de déterminer si les raisonnements réalisés par un SBC sont *corrects* et *complets* par rapport à la sémantique formelle du langage utilisé. Informellement, un raisonnement est dit correct lorsqu'il n'a pas conduit à déduire *plus* de connaissances que ce qui est prévu dans la sémantique formelle. De façon complémentaire, un raisonnement est dit complet lorsqu'il ne *manque aucune* des déductions prévues par la sémantique formelle. Ces vérifications et comparaisons permettent de s'assurer que deux SBC différents conçus pour un même usage ne conduisent pas à une représentation ou une production de connaissances à la signification divergente.

Le domaine de recherche regroupant les réflexions sur une conception des SBC qui tienne compte de leur usage se nomme l'*ingénierie des connaissances* (cf. [BFP98], [Cha05] et [Bac04]). Autour de ce dernier gravitent d'autres catégories de recherches telles que le traitement automatique des langues étudiant les moyens d'analyser le langage naturel ; les recherches terminologiques tournées vers la sémantique des mots et des relations qu'ils entretiennent, les recherches sur l'interaction homme-machine étudiant les échanges entre un système et son utilisateur et la fouille de données recherchant des méthodes efficaces d'extraction d'informations suffisamment pertinentes.

Une partie de l'ingénierie des connaissances s'appuie sur les langages définis et les résultats obtenus dans le secteur de la *RCR* pour développer des solutions mieux adaptées au Web Sémantique. L'une d'entre elles consiste à traduire tout ou partie d'un langage de description de métadonnées (ou d'annotations) en un langage de *RCR* afin d'exploiter le pouvoir d'expression de ce dernier pour décrire avec richesse et rigueur le contenu d'une ou plusieurs ressource(s) web et d'enrichir ce contenu par raisonnement automatisé. La syntaxe adoptée par les langages existants de métadonnées est relativement proche du modèle entité-relation qui peut être assez facilement représenté de manière graphique. Associer une représentation visuelle à ces langages de description contribue à rendre la phase de modélisation plus intuitive et moins laborieuse à l'utilisateur.

Comme le souligne [Sow87], au cours du siècle précédent, d'autres formalismes représentant les informations sous une forme graphique ont été proposés. En 1909, C.S. Peirce développa les premiers graphes, dits existentiels, pourvus d'une interprétation et d'heuristiques logiques. En 1959, les recherches de Tesnière dans le domaine de la linguistique conduisirent à la définition d'un formalisme graphique dont la forme est mieux adaptée aux exigences du langage naturel bien que l'expressivité reste similaire à celle des graphes de Peirce. Dans le domaine de l'intelligence artificielle en 1966, R. Quillian débute un travail sur les réseaux sémantiques avec l'objectif de modéliser les capacités de la mémoire humaine. En 1975, apparurent les frames et les scripts (Minsky, 75) qui sont une notation plus formelle des réseaux sémantiques et qui conduiront plus tard à la définition des logiques de description ([BCM$^+$03]). Citons aussi au passage le formalisme UML (norme ISO proposée par l'OMG), permettant une représentation illustrée de l'information, et couramment utilisé dans le secteur industriel. Il est utilisé pour modéliser la spécification d'une architecture ou des besoins d'un projet. En 1984, J. F. Sowa s'inspire en particulier des graphes de Peirce pour définir le formalisme des graphes conceptuels [Sow84] et le munir d'une sémantique logique (similaire à celle des graphes existentiels). En s'appuyant sur la théorie des graphes [Ber58], il définit des opérations de projection, généralisation, spécialisation à partir desquelles des raisonnements peuvent être réalisés.

Les réflexions menées dans le Web Sémantique sur la description du contenu des ressources ont conduit à la naissance de deux langages de description de métadonnées, proches du modèle entité-relation et pouvant par conséquent être assez aisément représentés de manière graphique : les langages *Ressource Description Framework* (en abrégé RDF [Bec04], créé en 1999) et *Topic Maps* (en abrégé TM [ISO00] datant de 2001).

Le langage RDF est l'aboutissement d'une initiative, proposée en 1994 par Tim Berners Lee, de formalisation des métadonnées. Il s'appuie sur une syntaxe XML qui peut être considérée comme un format de sérialisation de graphe. Un document RDF est un ensemble de triplets (sujet, prédicat, objet) qui représentent une ressource (le sujet) en attribuant une valeur (l'objet) à une de ses propriétés (le prédicat). Les triplets RDF s'inspirent de la structure sujet-verbe-objet des expressions les plus simples du langage naturel. Une relation dite de *subsomption* peut être définie entre deux documents RDF.

Lorsqu'elle est établie telle que *a subsume b*, elle signifie que la situation représentée par *b* est un cas particulier de celle représentée par *a*.

Il peut être nécessaire d'organiser le vocabulaire employé dans un document RDF sous la forme d'une ontologie rudimentaire. Pour ce faire, on dispose du langage RDFS (voir [BG04]) qui permet de définir des classes et des propriétés ; et de les structurer en utilisant une terminologie normalisée de primitives (les classes `Class`, `Property`, etc, et les propriétés `type`, `subClassOf`, `subPropertyOf`, etc). Cette terminologie présente l'avantage d'être intuitive et synthétique mais peut parfois être insuffisante. En 2003, un nouveau langage plus expressif que RDFS apparaît : le langage OWL. Il est issu des recherches portant sur les logiques de description et se présente sous la forme d'une extension au langage RDFS. OWL introduit de nouvelles primitives (`allValuesFrom`, `minCardinality`, etc) permettant de structurer sous une forme plus fine le vocabulaire utilisé à la modélisation. Un document OWL permet de définir des individus, des concepts et des propriétés sur ces concepts. Les individus représentent les entités du monde, un concept un ensemble d'entités et une propriété un ensemble de relations établies entre entités. Comme pour RDF, OWL introduit une relation de subsomption définie entre concepts et propriétés. Lorsqu'elle est établie entre deux concepts *a* et *b* telle que *a subsume b*, elle signifie que l'ensemble des entités représenté par *b* est inclus dans l'ensemble des entités associé à *a*.

Les langages RDF/S[3] et OWL disposent de sémantiques formelles définies en théorie des modèles (cf. [Hay04] en ce qui concerne RDF/S et [PSHH04] en ce qui concerne OWL). Les sémantiques formelles permettent de définir l'interprétation mathématique à accorder au contenu des documents et aux relations de *subsomption*. Ces langages se situent donc parmi les langages de représentation des connaissances offrant un support théorique robuste pouvant faire l'objet de théorèmes mathématiquement vérifiables.

Un nombre important d'outils d'édition, d'interrogation et de raisonnement ont été conçus pour les langages RDF/S et OWL. En ce qui concerne RDF, on peut citer l'outil Sésame [eA] et Oracle RDF [Ora] qui permettent d'entreposer du RDF de manière persistante et offrent des services d'interrogation appropriés (Sésame s'appuyant sur le langage standard d'interrogation SPARQL [W3C07]). OWL dispose de nombreux outils d'édition (tels que SWOOP [KPS+06] et Protégé [Sta07]) et d'outils de raisonnement qui s'appuient en général sur les logiques de descriptions (Racer [HM01], FaCT++ [TH06], etc).

Bien que RDF/S et OWL s'imposent à l'heure actuelle comme les formalismes de représentation des connaissances les mieux adaptés au Web Sémantique, il existe cependant des alternatives, dont une assez répandue dans le secteur industriel : le langage des Topic Maps (TM). A la différence du langage RDF qui se situe directement dans la

[3]Le sigle RDF/S signifie qu'on se réfère au langage RDF ou bien au langage RDF complété des primitives RDFS.

continuité des recherches réalisées sur le Web, le formalisme des TM est issu du secteur documentaire, voire terminologique puisqu'il doit sa création à un projet en 1995 de fusion automatique d'index et de glossaires provenant de différents documents électroniques. Les TM ont été définies pour structurer un ensemble de ressources (documents adressables) sous la forme de topics partageant des propriétés communes et d'associations indiquant des liens entre topics [ISO00]. Les topics peuvent être identifiés par des *noms* et caractérisés par des *occurrences* qui indiquent une information pertinente pour le topic. Deux associations spécifiques sont normalisées pour toute TM : l'association *classe-instance* et l'association *superclasse-sousclasse*. Ce langage dispose d'une syntaxe XML [Top01] qui en fait aussi un des langages candidats à une utilisation dans le cadre du web sémantique.

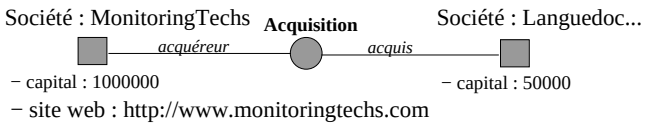


FIG. 1.2 – Une TM.

Les TM peuvent être représentées sous une forme graphique dont les segments indiquent le rôle joué par un topic dans une association et leurs extrémités représentent, sous la forme d'une carré et d'un disque, le topic et l'association concernés. La TM présentée dans la figure 1.2 illustre l'acquisition d'une société par une autre sous la forme d'un topic de classe `Société` et de nom `MonitoringTech` qui joue le rôle `acquéreur` dans une association de type `acquisition` dans laquelle le topic de nom `Languedoc Economic-Analyse And Monitoring Company` joue le rôle `acquis`. Deux occurrences sont associées au topic `MonitoringTechs` : la valeur `1000000` de type `entier` comme `capital`, et la valeur `www.monitoringtechs.com` de type `URL` comme `site web`. Ces occurrences peuvent s'interpréter comme le capital de la société MonitoringTechs et l'adresse web permettant de la contacter. Contrairement à ses concurrents directs, RDF/S et la famille OWL, le formalisme des TM n'a pas été muni d'une sémantique formelle et dans le secteur industriel, bien que certaines sociétés comme Mondeca ou Ontopia développent des outils de gestion des connaissances basés sur les TM, il semble qu'aucun outil complet de raisonnement[4] n'ait à ce jour été proposé pour ce langage.

La syntaxe et la sémantique des langages du Web Sémantique ont été réunies dans des documents dits *standards* qui sont le résultat d'importants consensus entre les secteurs industriels et scientifiques. Bien qu'un rôle fondamental soit attribué à ces langages

[4] c'est à dire un outil permettant d'offrir des services en raisonnement allant au delà de la simple interrogation et permettant de produire de nouvelles connaissances par application de règles d'inférence sur les connaissances de la base.

dans la conception des services "intelligents" de demain, ces documents standards ne présentent pas pour autant les algorithmes de raisonnement nécessaires à la réalisation de ces services. Ces langages nécessitent donc d'être *opérationalisés* soit directement, soit par d'autres langages pourvoyeurs de ces algorithmes provenant principalement du domaine de la *RCR* [5]. L'opérationalisation d'un langage par un autre peut être réalisée de différentes manières, dont une consiste à traduire les expressions du langage en celles de l'autre. Cette transformation est parfois sujette à respecter certaines contraintes ; notamment s'il est nécessaire de préserver la sémantique formelle des représentations (lorsqu'elles en sont pourvues) ou les rapports formels existant entre ces représentations (déduction, subsomption, projection, etc).

De nos jours, les métadonnées des ressources Web sont définies dans les anciens langages ; c'est à dire en utilisant le format hypertexte conçu pour le Web. Dans un avenir proche, il sera nécessaire d'harmoniser le contenu de ces métadonnées, c'est à dire l'ancien contenu structuré du Web, avec celui du Web Sémantique. Une éventuelle transition entre les métadonnées du Web vers celles du Web Sémantique pourrait être établie en incitant l'auteur ou le propriétaire des ressources annotées à formaliser son contenu dans un des langages plus riches prévus à cet effet. Il n'en demeure pas moins que l'établissement manuel de ces descriptions peut devenir laborieux et la richesse de la description dépendra du temps qui lui aura été consacré. Or, eu égard à l'empressement dont fait état la société et principalement les domaines soumis à forte concurrence (tels que les marchés, le monde du commerce, etc), il est probable que l'individu, plus enclin à privilégier la productivité au détriment de la qualité, produise des descriptions pauvres au regard des possibilités d'expression permises par ces langages. A cela s'ajoute la quantité considérable de métadonnées à améliorer induite par l'abondance des sites web (plus de 100 millions).

L'alternative à cette approche consiste à développer des méthodes pour déterminer automatiquement les informations importantes pour la modélisation et la structuration du contenu. Le domaine de l'ingénierie linguistique se concentre sur la partie littérale des contenus (afin de déterminer, par exemple, la fonction d'un mot relativement aux autres dans une phrase) et celui de l'analyse de l'image sur la partie graphique (pour déterminer la position d'un objet dans une image par rapport aux autres, par exemple). Des progrès suffisants dans ces domaines pourraient conduire à un procédé générique d'annotation de ressources :

- Un mécanisme d'annotation propose une structure pour chaque ressource identifiable sur le Web.
- En s'appuyant sur une correspondance établie entre la terminologie utilisée pour structurer le contenu de la ressource et le vocabulaire défini dans une ou plusieurs

[5] Dans la mesure du possible, car parfois, lorsque le pouvoir d'expression du langage est trop important les mécanismes permettant de l'opérationaliser n'existent pas ou sont inadaptés à l'usage pratique.

ontologies, cette structure est traduite en une expression d'un langage de représentation des connaissances et devient une *annotation*.

- Ces annotations, éventuellement enrichies par les capacités de raisonnement des SBC, sont regroupées en entrepôts de métadonnées.

L'intérêt de ce regroupement final réside en ce qu'il permet de centraliser, d'agréger en un emplacement et un format uniques des informations extraites de sources multiples dispersées sur le Web. Ces considérations autour de l'*agrégation de contenu* rejoignent alors les objectifs plus larges de l'*intégration de sources de données* (cf. [HR05]) qui consistent à offrir - en général par l'intermédiaire de systèmes de médiation - un accès *homogène* à une grande quantité d'informations qualifiées d'*hétérogènes* de par leurs emplacements, leurs structures, leurs formats, voire leurs langues et leurs sémantiques. La qualité de l'information provenant de sources aussi diverses doit être soumise à caution. En effet, il convient de s'interroger sur leur pertinence sous peine de diffuser des absurdités qui ne l'auraient pas été avec un accès conventionnel. Le système offrant cet accès peut alors faire appel à des mécanismes de contrôle pour détecter des incohérences ou jauger la qualité des connaissances décrites dans les annotations (en vérifiant les sources, la structure, ou encore la présence d'éléments interdits et obligatoires, etc).

L'ambition du Web Sémantique pourrait donc se résumer à l'amélioration de la qualité de la recherche d'information et de la communication inter-agents qu'offre actuellement le Web :

- en formalisant la fonction des *services Web* ou le contenu des *ressources Web* par des *métadonnées* ou *annotations* formulées en un langage de description suffisamment expressif, dont le vocabulaire peut être défini par une *ontologie* et dont l'*opérationalisation* par un *langage de représentation des connaissances* le munit de capacités de *raisonnement* (s'il ne l'est pas déjà) ;
- en développant davantage les *accès homogènes* au contenu *hétérogène* du Web par des procédés de *médiation* ou d'*agrégation de contenu* s'appuyant éventuellement sur des *processus d'annotation* pour structurer le contenu ;
- en qualifiant les connaissances considérées pour évaluer leur pertinence ;
- et en s'assurant que les procédés mis en place *passent à l'échelle* c'est à dire qu'ils sont applicables au Web dans son ensemble et résistent suffisamment à la charge d'informations induite par l'abondance des ressources.

Et pour remplir ces objectifs, les recherches s'articulent principalement autour des domaines suivants :

- *l'ingénierie des connaissances* qui étudie les moyens de représenter les connaissances humaines et d'automatiser les raisonnements sur ces connaissances en s'appuyant sur les résultat obtenus en *RCR* tout en les adaptant à un contexte d'utilisation précis ;
- *l'ingénierie linguistique* qui se consacre aux méthodes d'analyse du contenu littéral des ressources Web,

– *l'interaction homme-machine* qui étudie les conditions favorisant l'échange entre un système et son utilisateur (et inversement).

Le spectre des recherches réalisées s'étend à de nombreux autres secteurs tels que les bases de données, l'apprentissage automatique et les hypermedias. Les innovations dans ces domaines ont conduit à l'émergence de services commerciaux qui répondent aux besoins des organisations industrielles et administratives, de capitaliser et d'exploiter à un niveau sémantique le patrimoine immatériel que constituent les savoirs et savoir-faire des collectivités.

Contexte industriel

Mondeca est un éditeur de logiciels qui développe un outil de gestion des connaissances et des contenus : ITM (Intelligent Topic Manager). Les clients de Mondeca s'inscrivent dans des secteurs d'activité variés tels que la presse people (Hachette), la veille médiatique (PressIndex), le secteur législatif (Wolters Kluwer, Lexis Nexis, Veritas) et celui de la construction automobile (PSA). Ces entreprises peuvent réserver un usage interne à ITM pour organiser les ressources de leur intranet ou l'intégrer dans les solutions qu'elles commercialisent et destinent à un usage sur Internet.

ITM rassemble des connaissances de diverses natures (ontologie, thésaurus, taxinomie) et s'appuie sur une base de connaissances sur laquelle ont été construits différents services : indexation terminologique des documents à partir du thesaurus, gestion de la structure organisationnelle des documents, construction d'annotations contrôlée par l'ontologie de domaine, annotation semi-automatique de documents, navigation dans la base d'annotations, recherche dans la base d'annotations (cf. figure 1.3).

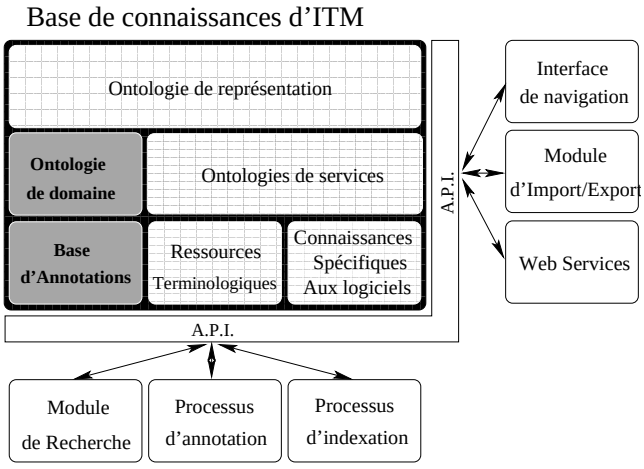


Fig. 1.3 – La base et les modules d'ITM.

Cette base de connaissances est structurée en 3 niveaux :

1. Le niveau méta contient l'ontologie de représentation. Il comporte toutes les primitives permettant de créer des ontologies de domaine pour chaque client de l'outil ITM. Ce niveau évolue très peu et est commun à tous les clients.
2. Au niveau modèle, des ontologies sont créées pour chaque client en utilisant les primitives de l'ontologie de représentation. On définit ici des types et des propriétés générales qui seront utilisées au niveau des instances. Plusieurs sortes d'ontologies sont généralement définies : une ontologie de domaine spécifique au client, des ontologies spécifiques "de services" (primitives de spécification de thésaurus, primitives de description d'une organisation logique des documents...) permettant la mise en oeuvre des services offerts par ITM.
3. Le niveau des instances contient des annotations sémantiques, thésaurus, et des descriptions organisationnelles construites en utilisant le vocabulaire conceptuel des ontologies du niveau modèle.

S'appuyant sur sa base de connaissances stockée dans un SGBD, ITM propose différents services d'acquisition et d'exploitation de ces connaissances.

L'acquisition des connaissances est réalisée par l'intermédiaire d'interfaces à base de formulaires "dynamiquement contrôlés" par l'ontologie correspondante au niveau modèle. Il existe par ailleurs un module d'annotation semi-automatique qui permet d'alimenter ITM avec des connaissances extraites de documents non structurés et semi-structurés sur la base de méthodes linguistiques [Ama06]. Enfin un module d'importation de connaissances représentées dans divers formats (XTM [ISO00], RDF [Bec04], OWL [MvH04], SKOS [MB08]) permet d'intégrer des connaissances issues d'autres outils ; et en particulier des ontologies de domaine construites à l'aide d'éditeurs spécialisés.

L'exploitation des connaissances peut se faire par navigation graphique dans le réseau des connaissances. Une méthode de recherche "full-text" permet d'explorer les libellés textuels mais aussi les documents textes (externes à ITM) associées aux annotations dans la base. Une recherche multi-critères est également proposée. Elle permet de réaliser des recherches fines en spécifiant à l'aide de formulaires les caractéristiques précises de l'annotation ou de l'élément recherché dans la base.

Cet ensemble de services constitue une API qu'ITM fournit en tant que service web.

Bien qu'étant un outil complexe de gestion des connaissances, ITM ne permet pas la mise en œuvre de raisonnements. Du point de vue des usages, des capacités de raisonnement seraient utiles à ITM pour (1) vérifier la cohérence des connaissances représentées dans la base et (2) enrichir ces connaissances en en déduisant de nouvelles à partir de règles d'inférence.

Travail de thèse

L'objectif de cette thèse est précisément d'améliorer la gestion des connaissances d'ITM en le munissant de capacités d'inférences et de contrôle.

Le **chapitre 2** présente en détail le langage inspiré des TM utilisé dans ITM pour représenter les connaissances et les modèles théoriques qui ont été définis pour donner au langage standard des TM un cadre formel. La fin du chapitre est consacrée à la proposition d'un modèle différent des précédents défini pour les besoins de la thèse et structuré selon son usage dans ITM.

La définition et l'opérationalisation des notions de cohérence et d'enrichissement, nécessitent de doter les connaissances d'ITM d'une sémantique formelle et de faire appel à un formalisme de représentation de connaissances qui soit (1) proche de celui d'ITM, c'est-à-dire des TM, (2) doté de mécanismes de raisonnement et (3) pourvu d'une sémantique formelle. Le formalisme des graphes conceptuels, et plus précisément la famille $\mathcal{SG}$ [BM02] basée sur les graphes conceptuels simples et leurs extensions, a été choisi pour répondre à ces besoins pour les raisons suivantes :

- Il existe une grande similarité entre les TM et les graphes conceptuels simples (SG) ; en effet ces deux formalismes sont des réseaux de connaissances qui représentent des entités et les relations qu'elles entretiennent.
- Les graphes conceptuels possèdent une sémantique formelle en logique des prédicats.
- La famille $\mathcal{SG}$ fournit différents types de connaissances : les faits représentés par des graphes conceptuels simples, les règles d'inférence, ainsi que les contraintes *négatives* pour interdire les "excès" de connaissances et *positives* pour en détecter les "manques". Les règles d'inférence et les contraintes sont candidates pour respectivement couvrir les besoins en enrichissement et en contrôle.
- Les raisonnements de la famille $\mathcal{SG}$ sont basés sur des opérations de graphe. Ils s'appuient directement sur les connaissances représentées, sans passer par un langage logique. Ceci fournit potentiellement des possibilités d'explication des raisonnements à l'utilisateur puisque les raisonnements peuvent être visualisés dans le formalisme qu'il connaît et directement sur les connaissances qu'il a définies. Les raisonnements sont adéquats et complets par rapport à la sémantique logique des connaissances représentées. L'adéquation signifie que s'il existe une projection entre un SG et un autre alors la traduction logique du premier est la conséquence logique de celle du second. La complétude signifie que si la formule logique attribuée à un SG est conséquence logique de celle attribuée à un autre SG alors il existe une projection du premier dans le second.
- Les algorithmes développés pour les graphes conceptuels, et en particulier pour la famille $\mathcal{SG}$, utilisent des techniques de combinatoire (notamment de théorie des graphes et de satisfaction de contraintes) qui les rendent particulièrement efficaces.

– La famille $\mathcal{SG}$ est implémentée dans CoGITaNT, une API de développement d'applications à base de graphes conceptuels, utilisable librement [Gen97].

Le **chapitre 3** présente les différents composants et services fournis par la famille SG. Ces derniers s'articulent principalement autour de deux notions : les graphes conceptuels simples (SG) et le mécanisme de projection d'un graphe dans un autre. Un SG est constitué de *sommets concepts* et de *sommets relation* typés et reliés entre eux par des arêtes étiquetées ; un sommet concept pouvant représenter une entité du monde identifiée ou non. Les types des sommets concepts et relation sont définis et forment un ordre partiel dans une ontologie rudimentaire nommée *support*. Le mécanisme de projection d'un graphe dans un autre consiste à déterminer les portions du graphe cible qui représentent des connaissances plus spécifiques que celles représentées par le graphe source.

Les notions de SG et de projection permettent de définir des composants plus expressifs et des opérations plus complexes tels que les règles, les contraintes positives, les contraintes négatives et les opérations qui leur sont dédiées. Les règles SG représentent des connaissances du type "si *hypothèse* alors *conclusion*" et sont définies à partir de deux graphes SG, l'un représentant la partie hypothèse et l'autre la partie conclusion de la règle. Les contraintes positives (resp. négatives) ont la même forme que celle des règles à ceci près que l'hypothèse et la conclusion sont respectivement qualifiées de condition et d'obligation (resp. d'interdiction). Leur signification est du type "si la *condition* est présente alors il doit (resp. ne doit pas) en être de même pour la partie *obligation* (resp. *interdiction*)".

Chaque composant (graphe SG, règle, contrainte) peut être traduit en une formule de la logique des prédicats du premier ordre. On s'appuie notamment sur cette traduction pour démontrer que l'opération de projection est adéquate et complète par rapport à la déduction logique. Les opérations dédiées aux règles et aux contraintes disposent aussi d'une sémantique logique.

L'opérationalisation du formalisme des TM par la famille $\mathcal{SG}$ pose le problème de l'interopérabilité entre ces deux formalismes en le sens qu'il est nécessaire de définir et choisir la correspondance la plus convenable parmi toutes celles qui existent théoriquement entre les deux langages. Pour ce faire, trois transformations possibles ont été étudiées et sont présentées au **chapitre 4** :

1. La première consiste à traduire les constituants élémentaires de la syntaxe du formalisme TM (i.e topic, association, occurrence, etc) en des types de concepts du support, définir un type de relation représentant le lien de dépendance qui lie un constituant à un autre (i.e. une occurrence est "liée" par ce lien à son topic, puisqu'elle ne peut exister sans lui) ; puis à traduire le contenu des trois niveaux (méta, modèle et instances) en un graphe de faits SG. Cette traduction permet de traiter les connaissances d'ITM en toute généralité mais elle exploite mal les spécificités de représentation du formalisme SG. En effet, le résultat ne présente

pas d'ordre sur les types, mélange dans le graphe de faits des connaissances des trois niveaux d'ITM et utilise des règles SG coûteuses pour gérer les propriétés des classiques relations *classe-instance* et *superclasse-sousclasse*. De plus les structures obtenues sont d'une trop grande complexité et trop mauvaise lisibilité pour être exploitables.

2. La seconde transformation réalise la traduction du méta modèle ITM dans un support SG et celle des deux niveaux inférieurs en graphe de faits SG. Cette traduction conduit à des problèmes similaires à ceux posés par la traduction précédente mais dans une moindre mesure : le résultat ne présente toujours pas d'ordre sur les types et mélange dans la base de faits des connaissances du niveau modèle et du niveau instances d'ITM, les connaissances du niveau méta étant cette fois-ci modélisées par le support. Cette approche utilise aussi des règles coûteuses pour gérer les relations *classe-instance* et *superclasse-sousclasse*.
3. La troisième consiste à plaquer le méta-modèle ITM sur le formalisme SG, traduire le niveau modèle dans le support et traduire le niveau instance en graphe de faits SG. Cette transformation peut en plus produire un jeu de règles et de contraintes pour prendre en compte la sémantique de certaines déclarations au niveau modèle. Elle n'est appliquée qu'à la portion d'ITM dédiée aux annotations ; cette portion étant considérée comme la seule concernée par la représentation de connaissances (le thésaurus contenant des connaissances terminologiques et les autres espaces étant dédiés à la gestion de contenu). Cette transformation présente aussi l'avantage d'être réversible. Le support résultant de cette transformation contient un type pour chaque type déclaré au niveau modèle. L'ensemble des types de concepts respecte l'ordre induit par l'association TM *classe-sousclasse* et les règles coûteuses des deux transformations précédentes ne sont plus utilisées.

C'est ce dernier choix qui convient le mieux à ITM.

L'opérationalisation des TM selon cette approche a conduit à la définition d'un service de validation et d'enrichissement d'annotations répondant aux besoins d'ITM en raisonnement et en qualification de ses connaissances. Ce service, présenté au **chapitre 5**, permet principalement de contrôler la cohérence de chaque connaissance ajoutée à ITM sous forme de TM et d'en déduire de nouvelles par inférence (l'ajout de règles et des contraintes est également pris en compte). Il maintient une copie, enrichie par application de règles d'inférence, d'une portion "cohérente" de la base d'annotations. Le choix de la portion "cohérente" est déterminé par l'ordre d'ajout des annotations à la base ; toute nouvelle annotation rendant la base incohérente est rejetée (accompagnée des raisons de son rejet). Ceci permet d'assurer la monotonie des raisonnements effectués sur cette base ; c'est à dire qu'en cas d'acceptation les déductions réalisées et les connaissances de la base ne sont plus remises en cause au cours des prochaines modifications de la base.

Ce service est fondé sur un mécanisme d'ajout contrôlé (cf. chapitre 5) élaboré à partir de la famille $\mathcal{SG}$. La base de connaissances SG du mécanisme d'ajout contrôlé se compose d'un support issu de la transformation du niveau modèle d'ITM ; un ensemble de règles d'inférence et un ensemble de contraintes, issues d'une part du niveau modèle d'ITM et d'autre part spécifiées par un expert du domaine ; et une base d'assertions B issue du niveau instance d'ITM et enrichie par les règles d'inférence.

A l'initialisation, la base B est vide. Chaque annotation ajoutée dans ITM est transformée en un graphe conceptuel A ensuite fourni au mécanisme d'ajout contrôlé. L'ajout contrôlé consiste à enchaîner 4 étapes : (1) le contrôle de A (qui vérifie la syntaxe et les types de A), (2) l'intégration de A à B, (3) l'enrichissement par les règles et (4) la validation de l'ajout (qui détecte des manques de connaissances en vérifiant les contraintes positives).

A la fin des trois premières étapes, on déclenche la vérification des contraintes négatives pour détecter les excès de connaissances. En cas de viol d'une contrainte les TM correspondant à la portion incohérente de la base sont envoyées à ITM. Les TM inférées ou incohérentes sont obtenues par application inverse de la transformation d'une TM en SG avant d'être retournées à ITM.

Ce service a été adapté aux exigences d'une utilisation concrète et a été mis en application sur des données concrètes dans le cadre d'un projet client (le projet PressIndex) et d'un projet de recherche (le projet Eiffel). Lors de la phase d'implémentation détaillée au **chapitre 6**, il a été tenu compte du fait que le volume important des bases gérées par ITM peut conduire à des problèmes de performances. Pour garantir des temps de réponse relativement convenables, les étapes de vérification des contraintes négatives, d'enrichissement et de validation sont réalisées de manière *incrémentale* ; c'est à dire que ces traitements s'appliquent à cibler la plus petite partie possible de la base.

Chapitre 2

Les Topic Maps

Le concept de Topic Map (TM) est apparu au début des années 90, dans les secteurs documentaire et terminologique, au cours d'un projet de création d'index et de glossaires à partir de documents dispersés sur le web. Un consortium international dirigé par M. Biezunski et S. R. Newcomb se met alors en place pour formaliser ce concept. Au début de l'année 2000, au même moment que la création du standard RDF, il propose une syntaxe concrète en XML nommée *XML Topic Maps* (XTM) (voir [Top01] et [HP02]) qui sera adoptée sous la forme d'une norme ISO [ISO00].

Par rapport à l'idée initiale prévue pour répondre à des besoins précis du secteur documentaire, le standard *XTM* est destiné à un usage beaucoup plus général : il est défini pour décrire toute chose pouvant être l'objet d'un discours sous une forme suffisamment intuitive et structurée pour être exploitable à la fois par l'homme et la machine. Une TM est composée de *topics* caractérisés par des *occurrences* et d'*associations* qui relient ces topics entre eux. Les topics représentent des sujets, les occurrences les caractéristiques des sujets représentés et les associations les rapports qu'entretiennent entre eux ces sujets. D'un point de vue des usages, le formalisme des TM permet essentiellement de structurer un ensemble de ressources identifiables sur le Web.

Conjointement à l'élaboration du format XTM en tant que syntaxe concrète de référence, différents travaux de recherche proposent une syntaxe abstraite afin de représenter une TM dans un modèle mathématique ou de la manipuler en machine. Une première proposition de syntaxe abstraite est réalisée par [BN01] peu après l'adoption du standard. Elle propose de modéliser une TM par un graphe (au sens théorie des graphes) afin d'énoncer des conditions de fusion entre deux TM. Un autre travail réalisé au cours de l'année 2002 et décrit dans [AdMRV02] s'appuie aussi sur la théorie des graphes et propose, cette fois, de modéliser les TM par des hyper-graphes pour les organiser en niveaux.

Au cours de l'année 2006, une version révisée du standard XTM est proposée sous le

nom de *XTM 2.0*. Bien qu'elle n'introduise que des modifications mineures et ne soit pas encore adoptée comme syntaxe de référence, elle commence à être employée. Comme pour le premier format, certains travaux sont menés pour lui fournir une syntaxe abstraite. Le format TMDM proposé dans [GM06] en est une qui se rapproche des structures de données et sur laquelle s'appuient d'autres propositions parmi lesquels le modèle *Q* [Gar05a] et les *Subject Maps* [DNB06]. Ces deux formalismes sont employés pour représenter les connaissances décrites dans une TM et disposent d'un langage d'interrogation.

Ce chapitre commence par une présentation informelle du formalisme des TM et de sa signification. Ensuite, la section 2.2 décrit les deux versions existantes du standard *XTM*. La section 2.3 suivante présente les syntaxes abstraites proposées pour chacune des versions du standard. La dernière section décrit un des premiers résultats de la thèse : une syntaxe abstraite adaptée au langage de représentation des connaissances employé dans le logiciel ITM de Mondeca dont la plus grande partie est fondée sur le standard *XTM* ; l'autre ayant été créée pour satisfaire des besoins précis en représentation auxquels le standard ne répondait pas.

2.1 Présentation informelle

D'une façon générale, une TM modélise un *domaine* qui peut être soit abstrait (une idée), soit concret (une situation du monde réel). Le domaine se définit par un ensemble d'*entités* qui présentent des *caractéristiques individuelles* et entretiennent entre eux des *rapports* particuliers au sein desquels peut être identifiée la *participation* d'une entité à l'un de ces rapports. La figure 2.1 illustre un exemple de domaine qui présente dans la portion grise des entités du web (qualifiées de ressources adressables) et dans la portion blanche les entités du monde réel.

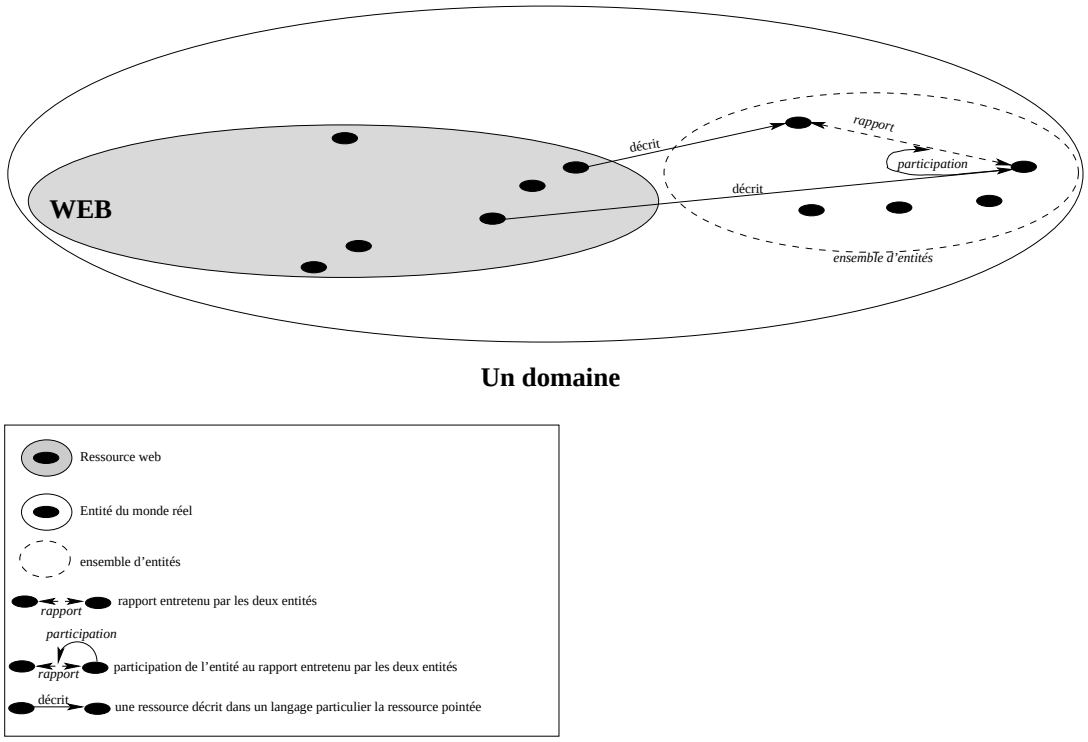


FIG. 2.1 – Un exemple de domaine.

Une TM est composée de *topics* et d'*associations* qui relient ces topics entre eux. Les topics sont instances de classes, peuvent être identifiés par des *noms* et caractérisés par des *occurrences*. Les associations sont typées et identifient le *rôle* joué par chaque topic dans une association. Deux associations spécifiques sont normalisées pour toute TM : l'association *superclasse-sousclasse* et l'association *classe-instance*. Tout élément association, occurrence ou nom de topic est défini au sein d'une sorte d'espace de nom : le *domaine de validité* (en anglais : *scope*) dont la signification précise est volontairement laissée libre, bien qu'il en existe une, présentée en partie 2.1, généralement admise comme standard par la communauté TM.

Le `topic`

L'entité que représente un topic est appelée le *sujet*. Un sujet peut être :
- un autre topic de la TM courante : dans ce cas on dit que le premier réifie le second,
- une ressource (une page web par exemple),
- une entité dont la définition est donnée par des sources indicatives d'infor ma tion.

La différence entre une ressource et une source indicative d'information tient en ce que le contenu de cette dernière permet de définir aussi précisément que possible le sujet représenté par le topic ; alors que la ressource (son support et son contenu) *est* le sujet représenté par le topic. On n'a qu'une idée imprécise de ce qui est défini par une source indicative d'information car en général le contenu de telles sources est hétérogène et décrit partiellement le sujet.

Lorsqu'une ressource localisable sur le Web par une URI *est* ou *définit* sans ambiguïté le sujet d'un topic, elle est qualifiée de *Published Subject Indicator* souvent abrégé par l'acronyme *PSI*. Le PSI permet alors d'identifier précisément le topic.

La figure 2.2 représente les significations possibles d'un topic. Le topic qui représente la ressource web d'adresse "`http://`www.jazzmanouche.com/bio1.html" représente la page web localisée à cette adresse. Par contre, si on considère le document "`http://`www.lesmanouches.com/bio-django.html" en tant que source indicative d'information pour un topic alors tout porte à croire que ce dernier représente l'individu "Django Reinhardt". Dans la suite, on désignera par *sujet référencé* un élément ressource ou une source indicative d'information.

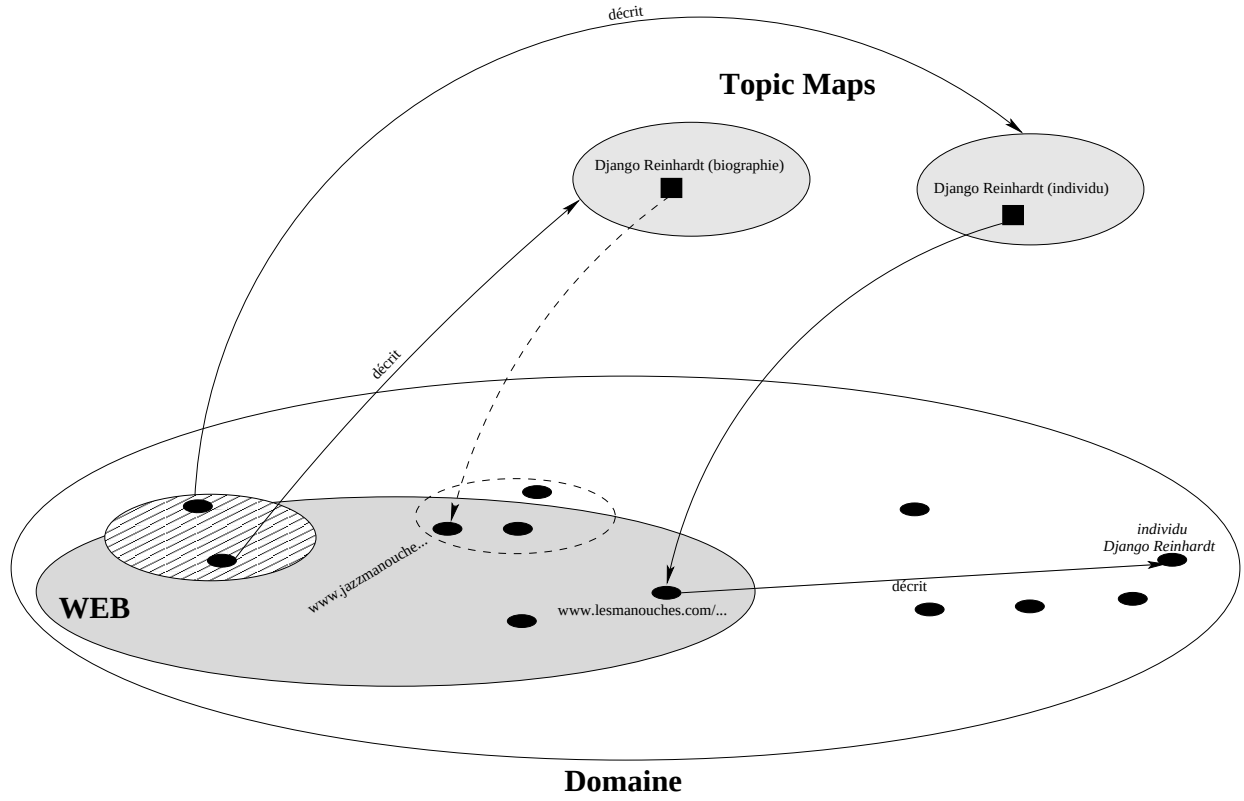


FIG. 2.2 – Deux topics : l'un représentant une ressource et l'autre une entité.

Un topic peut porter un ou plusieurs noms principaux qui correspondent à la désignation littérale communément admise de l'entité représentée. L'attribution d'un nom au topic peut être réalisée au sein d'un domaine de validité. A chaque nom principal peuvent être associées une ou plusieurs de ses *variations* (par exemple, l'acronyme ou une forme réduite pour le nom).

Un topic peut comporter des occurrences qui représentent les caractéristiques individuelles de l'entité représentée par le topic. Une occurrence peut ainsi désigner une

ressource ou représenter la valeur d'une donnée d'un certain type (entier, chaîne de caractères, date ou autre type primitif de donnée).

La figure 2.3 présente deux topics portant des occurrences. Le topic de nom `MonitoringTechs` représente la société de même nom et possède deux occurrences représentant respectivement le capital et le site web de la société. L'autre topic présenté dans cette figure porte les deux noms `Languedoc ...` et l'autre `Société ...` ; chacun d'entre eux étant respectivement définis dans un domaine de validité identifié par `english` ou `français`. A chaque nom est associé son acronyme `L.E.A.M.C.` et `S.A.V.E.L.` sous la forme d'une variation.

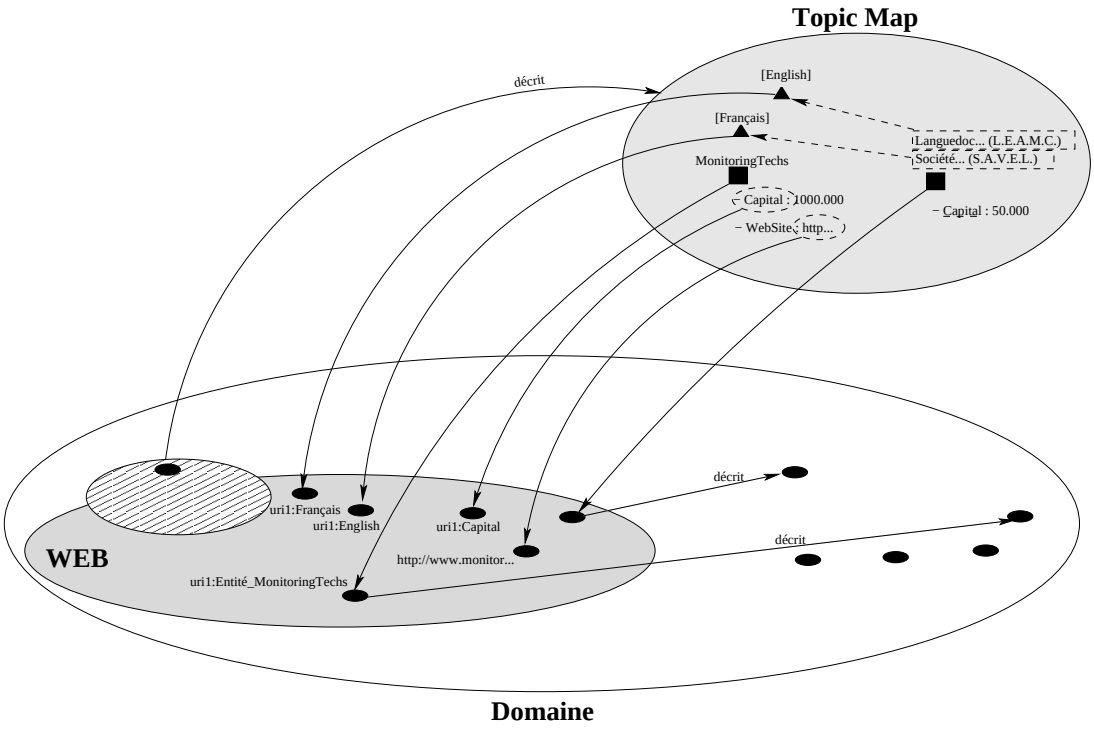


FIG. 2.3 – Un topic et ses occurrences.

Les `associations`

Les associations au sein desquelles les topics jouent un rôle permettent de représenter les rapports qu'entretiennent les entités représentées par ces topics. Pour représenter la participation d'une entité dans le rapport qu'elle entretient avec les autres, un *rôle* est assigné au topic représentant l'entité en question au sein de l'association. On dit que le

topic joue ce rôle dans cette association ; un topic pouvant jouer un même rôle dans différentes associations. La figure 2.4 présente une association représentant une acquisition de société dans laquelle les topics de nom `MonitoringTechs` et de nom `Languedoc ...` y jouent respectivement le rôle d'`acquéreur` et d'`acquis`.

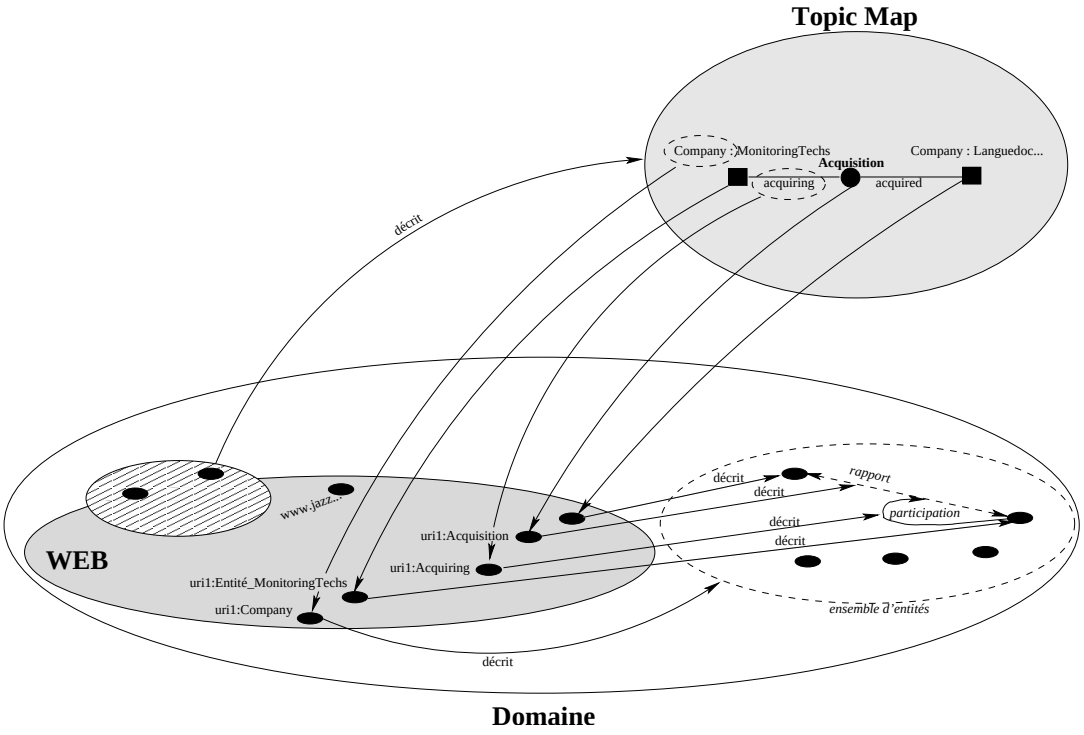


FIG. 2.4 – Une association entre deux topics.

Les `classes`

On considère maintenant les trois ensembles E, R et C contenant respectivement les entités du domaine, les rapports qu'entretiennent ces entités et les caractéristiques particulières à chaque entité. Un sous-ensemble d'un de ces trois ensembles E, R et C peut être représenté par un topic ou une source indicative d'information. Le topic représentant un sous-ensemble de E est qualifié de *classe* de topic, celui représentant un sous-ensemble de R ou C est respectivement appelé un *type d'association* ou un *type d'occurrence*[1]. D'une manière générale, on désignera par *type* un topic *classe*, *type*

[1]Il faut noter que la notion de type d'occurrence est indépendante du type primitif de sa valeur lorsque l'occurrence représente une valeur.

d'association ou *type d'occurrence*.

En TM, l'appartenance d'un élément x du domaine (une entité, un rapport ou une caractéristique) à un sous-ensemble X de E, R ou C est représentée en associant le *type* représentant X au topic, à l'association ou à l'occurrence représentant x ; ce dernier (ou cette dernière) portant dans la suite le nom d'*instance*. Techniquement, l'*instance* en question est reliée à son *type* par une association de type *classe-instance* dans laquelle elle joue le rôle *instance* et son type joue un rôle *classe*. Dans les figures, cette association n'est pas représentée. Pour plus de lisibilité, la classe est indiquée avant le nom du topic, le type d'association au dessus ou en dessous du sommet association et le type d'une occurrence précède sa valeur. La figure 2.4, par exemple, présente des topics instances de la classe `Company` et une association de type `Acquiring`. Dans la figure 2.3, les occurrences sont de type `Capital` ou `WebSite`.

Le formalisme des TM dispose d'une association *superclasse-sousclasse* qui permet d'établir des relations de spécialisation entre classes ou types. Lorsqu'une telle association est définie entre deux topics, celui jouant le rôle de *sousclasse* représente un sous-ensemble de E, R et C plus *spécifique* que celui représenté par le topic jouant le rôle de *superclasse*. La définition exacte de *spécifique* est volontairement laissée libre ; bien qu'elle semble s'aligner sur celle d'*inclusion* ensembliste.

La notion d'identité

D'une façon générale, lorsqu'un langage de représentation des connaissances vérifie la *unic name assumption* (UNA) cela signifie que deux symboles différents de ce langage désignent deux entités différentes dans le domaine modélisé. Le langage des TM ne respecte pas précisément la *UNA* généralement admise. En effet, deux éléments différents d'une TM n'en représentent pas forcément deux de différents dans le domaine modélisé. Par défaut, on ne connaît rien de leur identité ou différence. Par contre, ils sont dits *identiques* lorsqu'il a été précisément établi qu'ils représentent bien le même élément du domaine. Bien que l'interprétation de l'identité entre deux éléments d'une TM reste relativement libre, [BN01] énonce des conditions pour l'établir qui sont en général approuvées par la communauté TM. Ces conditions portent sur le PSI ou le nom du topic.

Deux topics sont dits *identiques* s'ils portent au moins un même PSI ou un même nom au sein d'un même domaine de validité, s'ils représentent des ressources *équivalentes* ou si l'entité que l'un d'eux représente est définie par au moins une source indicative d'informations équivalente à l'une de celles définissant l'entité représentée par l'autre topic ; l'interprétation de la notion d'*équivalence* demeurant libre.

Deux occurrences désignent une même caractéristique si elles ont des types et des domaines de validité identiques, des ressources ou valeurs équivalentes et appartiennent à des topics identiques ; l'interprétation de la notion d'équivalence demeurant libre ici aussi.

Deux associations désignent un même rapport à condition qu'elles partagent des types et des domaines de validité identiques ; et que chaque topic jouant un rôle dans l'une des associations soit identique à l'un de ceux jouant ce même rôle dans l'autre association.

Lorsque le domaine de validité d'un nom de topic, d'une occurrence ou d'une association n'est pas précisé, il est dit "non contraint". Un nom défini dans un tel domaine identifie le topic qui le porte de façon unique dans tous les domaines de validité. En ce qui concerne les associations ou les occurrences définies dans un domaine "non contraint", elles ne sont plus tenues de respecter la condition sur le domaine de validité pour respectivement vérifier l'identité avec un autre élément de même nature. En revanche, rien ne permet d'affirmer la différence ou l'identité entre deux noms, occurrences ou associations qui sont définis au sein de domaines de validité différents.

2.2 Syntaxes concrètes

Cette section présente les deux versions du format *XTM* : le standard et la nouvelle version en cours d'élaboration.

2.2.1 Standard *XTM 1.0*

Le formalisme des TM dispose d'une syntaxe concrète définie en XML et standardisée sous forme de norme ISO [ISO00] : le format XTM. Un document XTM décrit une TM sous la forme d'éléments XML `<topic>` et `<association>` placés sous la racine `<topicMap>` du document. Tous les éléments sont identifiés par un attribut `id`.

L'élément `<topic>` contient au plus un élément `<subjectIdentity>` qui identifie l'entité représentée par le topic. Ce dernier contient les éléments suivants choisis selon le type d'entité à représenter :

- au plus un élément `<resourceRef>` permet de désigner la ressource que représente le topic ;
- éventuellement plusieurs éléments `<topicRef>` désignent les topics réifiés par le topic ;
- éventuellement plusieurs éléments `<subjectIndicatorRef>` établissent une référence vers chaque source indicative d'information.

L'élément `<topic>` peut présenter plusieurs éléments `<baseName>` qui comportent une chaîne de caractères introduite par `<baseNameString>` et encodent le nom principal attribué au topic. Les variations du nom principal (par exemple son acronyme, ou sa forme réduite, abrégée) sont appelées *variant*. En XTM, l'élément `<baseName>` comporte éventuellement des éléments `<variant>` composés de la chaîne de caractère qui encode le nouveau nom et d'un élément `<parameters>` qui permet de spécifier des informations additionnelles de traitement en cas de fusion de TM (voir [BN01]). Un `<variant>` peut lui même comporter d'autres `<variant>`.

Plusieurs éléments `<occurrence>` peuvent apparaître au sein d'un élément `<topic>`. La ressource de l'occurrence est spécifiée par l'élément `<resourceRef>` et la valeur par `<resourceData>`.

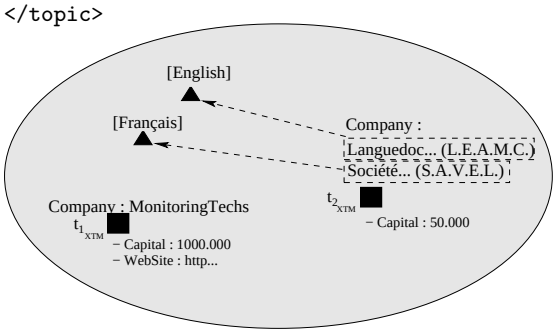
Un élément `<association>` sous la racine `<topicMap>` d'un document *XTM* contient au moins un élément `<member>` qui identifie les topics ou *sujets référencés* concernés par l'association et spécifie leur rôle par au plus un élément `roleSpec`.

Les éléments `<baseName>`, `<association>` ou `<occurrence>` peuvent contenir un élément `<scope>` qui décrit le domaine de validité du nom de topic, de l'association ou de l'occurrence en question.

Les éléments `<topic>`, `<association>` ou `<occurrence>` peuvent contenir un élément facultatif `<instanceOf>` (ou plusieurs chez le topic) qui permet d'attribuer un type au topic, à l'association ou à l'occurrence en question en établissant une référence vers le topic ou la source indicative d'information décrivant le *type*.

L'attribution d'un type peut aussi être décrite sous la forme d'une association de type *classe-instance* au sein même du document *XTM* et les types peuvent être organisées en utilisant des associations de type *superclasse-sousclasse*. Le type de ces associations est identifié par des *URI* réservées. Une association de type *classe-instance* relie un élément de rôle `instance` (topic, association ou occurrence) à un topic ou *sujet référencé* représentant le type ou la classe de l'élément en question. L'intérêt de cette solution par rapport à l'utilisation de `<instanceOf>` réside en la possibilité de définir un domaine de validité pour cette association (introduit par l'élément `<scope>`). L'association *superclasse-sousclasse* est définie entre deux topics ou *sujet référencé*, l'un de rôle *classe* et l'autre de rôle *sousclasse*.

```
<topic id="t1_XTM">
  <instanceOf>
    <subjectIndicatorRef
xlink:href="uri1:Company" />
  </instanceOf>
  <subjectIdentity>
    <subjectIndicatorRef xlink:href=
        "uri1:Entité\_MonitoringTechs" />
  </subjectIdentity>
  <baseName>
    <baseNameString>
MonitoringTechs</baseNameString>
  </baseName>
  <occurrence>
    <instanceOf>
      <subjectIndicatorRef xlink:href=
          "uri1:Capital" />
    </instanceOf>
    <resourceData>1000000</resourceData>
  </occurrence>
  <occurrence>
    <instanceOf>
      <subjectIndicatorRef xlink:href=
          "uri1:WebSite" />
    </instanceOf>
    <resourceData>
      http://www.monitoringtechs.com
    </resourceData>
  </occurrence>
</topic>
```



Topic Map

```
<topic id="t2_XTM">
  <instanceOf>
    <subjectIndicatorRef xlink:href="uri1:Company" />
  </instanceOf>
  <subjectIdentity>
    <subjectIndicatorRef xlink:href="uri1:Entité\_SAVEL"
  </subjectIdentity>
  <baseName>
    <scope>
      <subjectIndicatorRef xlink:href="uri1:Français" />
    </scope>
    <baseNameString> Société d'analyse et de veille
        économique du Languedoc </baseNameString>
    <variant>
      <variantName>
        <resourceData>S.A.V.E.L.</resourceData>
      </variantName>
    </variant>
  </baseName>
  <baseName>
    <scope>
      <subjectIndicatorRef xlink:href="uri1:English" />
    </scope>
    <baseNameString> Languedoc Economic Analyse
        And Monitoring Company </baseNameString>
    <variant>
      <variantName>
        <resourceData>L.E.A.M.C</resourceData>
      </variantName>
    </variant>
  </baseName>
  <occurrence>
    <instanceOf>
      <subjectIndicatorRef xlink:href="uri1:Capital" />
    </instanceOf>
  <resourceData>50000</resourceData>
  </occurrence>
</topic>
```

FIG. 2.5 – Un exemple de deux topics t_{1XTM} et t_{2XTM} représentés en XTM.

```
<association id="a_XTM">
  <instanceOf>
    <subjectIndicatorRef xlink:href="uri1:Acquisition" />
  </instanceOf>
  <member>
    <roleSpec>
      <subjectIndicatorRef
xlink:href="uri1:Acquiring" />
    </roleSpec>
    <topicRef xlink:href="t1_XTM" />
  </member>
  <member>
    <roleSpec>
      <subjectIndicatorRef
xlink:href="uri1:Acquired" />
    </roleSpec>
    <topicRef xlink:href="t2_XTM" />
  </member>
</association>
```

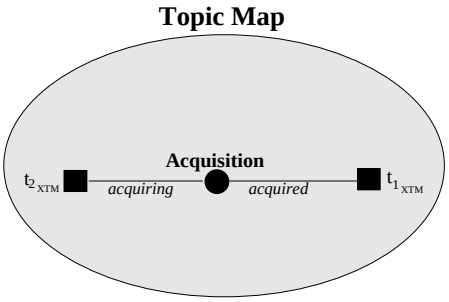


FIG. 2.6 – L'exemple d'une association a_{XTM} représentée en *XTM*. t_{1XTM} et t_{2XTM} désignent les topics présentés à la figure 2.5.

2.2.2 Format *XTM 2.0*

Le format *XTM 2.0* ([13206]) est une version quelque peu améliorée du format *XTM 1.0*. Bien que cette version ne soit pas encore standardisée, elle commence à être utilisée.

En ce qui concerne l'identification des sujets, les éléments `<resourceRef>` et `<subjectIndicatorRef>` sont respectivement remplacés par les éléments `<subjectLocator>` et `<subjectIdentifier>`; tandis que l'élément `<subjectIdentity>` est supprimé du format. D'une façon générale, dans cette version du format, seul le topic et l'occurrence sont en droit d'identifier une ressource. De ce fait, l'élément `<scope>` et l'élément `<role>` qui remplacent l'élément `<member>` ne comportent plus que des éléments `topicRef`. Dans la version précédente, ils pouvaient établir une référence vers une ressource ou une source indicative d'information par l'intermédiaire des éléments `<resourceRef>` et `<subjectIndicatorRef>` qui ont à présent disparu.

En ce qui concerne les noms de topics, ils peuvent en *XTM 2.0* être instance d'un type introduit pour distinguer différentes désignations littérales possibles d'un sujet. Par exemple, si le topic représente en tant que ressource un document biographique sur Django Reinhardt intitulé "La vie de Django", on peut soit lui associer un nom principal "Biographie de Django Reinhardt", soit le nom "La vie de Django" en typant le nom à l'aide d'un topic "Titre du Document". Quant à l'élément `basename`, il a été remplacé par l'élément `<name>` et `<baseNameString>` par `<value>`. De plus, on ne peut plus proposer une variation d'une variation de nom (l'élément `<variant>` ne peut plus

être le père d'un autre élément `<variant>`), on peut spécifier le domaine de validité d'une variation de nom (l'élément `<scope>` remplace `<parameters>` dans `<variant>`) et l'élément `<variantName>` devient `<resourceRef>` ou `<resourceData>` selon la nature du libellé.

En ce qui concerne le typage, les topics, occurrences, associations et rôles doivent obligatoirement avoir un type qui ne peut être qu'un topic. Dans la version précédente, il était facultatif et pouvait être défini à partir d'une ressource ou d'une source indicative d'information. Le type est spécifié par un élément `<type>` (dans la version précédente il l'était par un élément `<instanceOf>` ou `<roleSpec>`), à part chez les topics qui conservent l'élément `<instanceOf>`.

La réification d'une TM, d'un nom de topic, d'une occurrence, d'une association ou d'un rôle par un topic est réalisée par un attribut `reifier` ajouté aux éléments `<topicMap>`, `<name>`, `<variant>`, `<occurrence>`, `<association>` et `<role>`. Cet attribut établit une référence vers le topic réifiant l'élément qui le possède. Toujours pour faciliter la réification, l'attribut `id` a été supprimé au profit d'un élément `itemIdentity` contenu par tous les éléments *XTM 2.0* (sauf lui-même). Pour réifier un de ces éléments, il suffit de créer un topic dont l'élément `<subjectLocator>` contient la valeur de `itemIdentity` dans l'élément en question.

La précédente version du formalisme ne permettait pas de spécifier le type de la valeur que représentait une occurrence c'est pourquoi l'attribut `datatype` a été ajouté à l'élément `resourceData`.

Il suffit d'effectuer des modifications mineures dans leur syntaxe pour que les exemples de topics $t_{1_{XTM}}$, $t_{2_{XTM}}$ donnés en *XTM 1.0* et présentés dans la figure 2.5 vérifient les contraintes du *XTM 2.0*. La version de chacun d'entre eux en *XTM 2.0* est intégralement présentée en annexe à la section A.6. Ils sont désignés dans la suite par $t_{1_{XTM2}}$, $t_{2_{XTM2}}$. Pour les obtenir, on supprime l'élément `subjectIdentity` tout en conservant son contenu, puis on remplace les éléments `baseNameString`, `variantName`, `subjectIndicatorRef` et `resourceRef` respectivement par les éléments `value`, `resourceData`, `subjectIdentifier` et `subjectLocator`. L'élément `instanceOf` est remplacé par `type` dans les éléments `occurrence`. L'élément `baseName` devient `name`.

En ce qui concerne l'association a_{XTM} présentée en figure 2.6, elle vérifie la syntaxe *XTM 2.0* si on remplace `instanceOf` par `type` et ce dans l'élément `association` aussi bien que dans `member`, puis ensuite `member` par `rôle`. Sa version en *XTM 2.0* est désignée par a_{XTM2}.

2.3 Syntaxes abstraites pour *XTM*.

Les formats *XTM* précédents ont été conçus pour fournir aux TM une syntaxe concrète adaptée à l'échange. L'exploitation de ces connaissances est néanmoins diffi-

cilement réalisable tant qu'elles restent décrites en XML. Afin de les traiter plus efficacement, diverses syntaxes abstraites - parfois proches des structures de données - ont été proposées. Certaines s'appliquent à conserver la structure du document XTM (voir 2.3.1 et 2.3.2), d'autres représentent l'information à partir de tuples (voir 2.3.3) ou s'inspirent de la théorie des graphes (voir 2.3.4 ou 2.3.5).

2.3.1 Le modèle de données des Topic Maps (*TMDM*).

TMDM [GM06] permet de représenter un document *XTM 2.0* dans un format proche des structures de données facilement implémentable en machine.

Le modèle *TMDM* comporte 9 classes structurées en une hiérarchie permettant l'héritage d'attributs. La figure 2.7 présente ces classes. La classe au sommet se nomme `TopicMapConstruct` et représente la super-classe de tous les constructeurs du modèle TMDM. Sa sous-classe directe est la classe `Reifiable` qui représente la super-classe des constructeurs qui sont réifiables par au plus une instance de la classe `Topic`, c'est à dire un topic au sein d'une TM. Les 6 constructeurs réifiables sont `AssociationRole`, `TopicName`, `Association`, `Occurrence`, `Variant`, `TopicMap`.

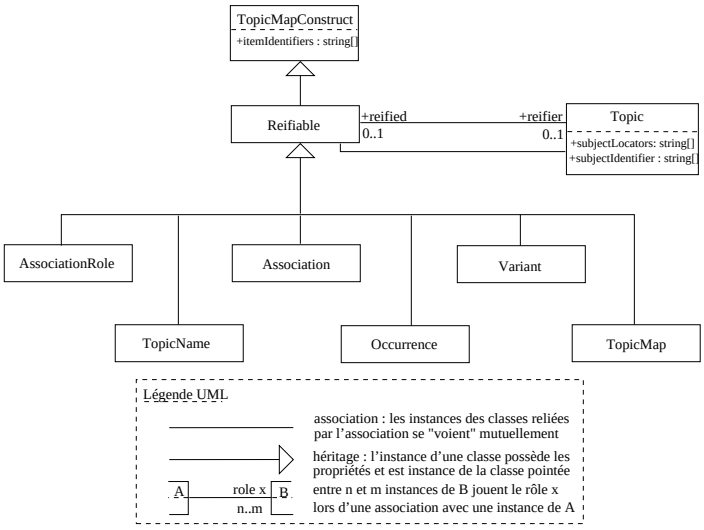


FIG. 2.7 – Les neuf classes du modèle TMDM.

En TMDM, un document *XTM 2.0* est représenté par une instance de la classe `TopicMap`. La classe `TopicMap` contient en tant qu'attributs des instances des classes `Topic` et `Association`.

Le format TMDM préserve la structure en arbre du document XML. D'une façon

générale, à chaque élément du document *XTM 2.0* correspond en TMDM une instance d'une des neuf classes précédentes. Lorsqu'un élément XML est contenu dans un autre élément, l'instance associée au premier est un attribut de l'instance associée au second. La section A.1 présente en détail le format TMDM en s'appuyant sur un document *XTM 2.0* composé des topics $t1_{XTM2}$, $t2_{XTM2}$ et de l'association a_{XTM2} donnés en exemple à la section 2.2.2.

2.3.2 Les Subject Maps

[BS05] présente un langage de représentation des connaissances appelé *Subject Maps* (SM) qui est actuellement en cours de standardisation en tant que norme ISO [DNB06]. Ce langage dispose d'un ensemble de primitives de navigation pouvant constituer la base d'un langage d'interrogation. Afin de se positionner par rapport aux TM, une correspondance est établie dans [DNB06] entre les structures du format *TMDM* et celles des *SM*. Cette correspondance n'est pas présentée dans cette section mais un exemple détaillé de représentation en *SM* d'une instance de *TMDM* est donné en annexe à la section A.2.

Une SM est un ensemble fini de *Subjects Proxies* (SP) ; ces derniers étant eux-mêmes des ensembles de *propriétés*. Chaque propriété est un couple clé-valeur $(k, v) \in \mathcal{L} \times \mathcal{V}$ avec $\mathcal{V}$ un ensemble de valeurs qui contient un ensemble $\mathcal{L}$ de libellés. Un *SP* peut contenir plusieurs propriétés partageant une même clé tant que leurs valeurs sont différentes. L'ensemble de toutes les propriétés définies sur $\mathcal{V}$ et $\mathcal{L}$ est désigné par $\mathcal{P}$, l'ensemble de tous les *SP* est $2^{\mathcal{P}}$ et $\mathcal{M}$ désigne l'ensemble de toutes les *SM*.

Une fonction *lib* partielle et injective, définie de $2^{\mathcal{P}}$ dans $\mathcal{V}$, retourne le libellé identifiant un *SP* donné. Cette fonction est étendue aux valeurs en considérant que pour toute valeur $v \in \mathcal{V}$ $lib(v) = v$.

Les deux relations particulières *sub* et *isa*

Les deux relations *sub* et *isa* sont toujours définies relativement à une *SM* m donnée. Lorsque sub_m met en relation deux *SP* x et y vérifiant $x\ sub_m\ y$, on dit qu'ils sont dans une relation *classe/sous-classe*, que x est *sous-classe* de y ou encore que y est *super-classe* de x. La relation sub_m est réflexive et transitive pour toute *SM* m. Lorsque deux *SP* a et b vérifient $a\ isa_m\ b$, on dit soit que a est l'*instance* de b, soit que b est le *type* de a. La relation isa_m est anti-reflexive, de plus si $y\ sub_m\ x$ et $z\ isa_m\ y$ alors $z\ isa_m\ x$ pour tout *SP* $x\ y\ z$.

Correspondance entre TM et SM

Une instance de *TMDM* qui représente un document *XTM 2.0* est représentée en *SM* par un *SP*.

$$tm = \left\{ \begin{array}{l} (topic, lib(t_1)) \\ (topic, lib(t_2)) \\ (topic, lib(comp)) \\ (topic, lib(inst)) \\ (topic, lib(class)) \\ (topic, lib(acq_{ing})) \\ (topic, lib(acq_{ed})) \\ (topic, lib(acq)) \\ (topic, lib(c-i)) \\ (topic, lib(class)) \\ (topic, lib(cap)) \\ (topic, lib(web)) \\ (topic, lib(eng)) \\ (topic, lib(fr)) \\ (topic, lib(tn)) \\ (association, lib(a_1)) \\ (association, lib(a_2)) \\ (association, lib(a_3)) \end{array} \right\}$$

$$t_1 = \left\{ \begin{array}{l} (topic\,name, lib(name_1)) \\ (occurrence, lib(occ_{web_1})) \\ (occurrence, lib(occ_{cap_1})) \\ (role\,played, lib(r_{acq_{ing}})) \\ (role\,played, lib(r_{inst_1})) \\ (subject\,identifier, lib(sub_1)) \\ (item\,identifier, lib(id_1)) \\ (parent, lib(tm)) \end{array} \right\}$$

$$a_1 = \left\{ \begin{array}{l} (role, lib(r_{acq_{ing}})) \\ (role, lib(r_{acq_{ed}})) \\ (scope, U) \\ (type, lib(acq)) \\ (parent, lib(tm)) \end{array} \right\}$$

$$r_{acq_{ing}} = \left\{ \begin{array}{l} (player, lib(t_1)) \\ (type, lib(acq_{ing})) \\ (parent, lib(a_1)) \end{array} \right\}$$

FIG. 2.8 – Quatre *SP* tm, t_1, a_1 et $r_{acq_{ing}}$ représentant respectivement une TM ; un topic et une association de cette TM ; et un des rôles joués dans l'association.

Par exemple, le *SP* tm présenté à la figure 2.8 représente l'instance de la classe `TopicMaps` du modèle *TMDM* représentant le document composé des topics $t_{1_{XTM2}}$ (correspondant à MonitoringTechs), $t_{2_{XTM2}}$ (correspondant à S.A.V.E.L.) et de l'association a_{XTM2}. Les éléments associés à la clé *topics* et ceux associés à la clé *associations* correspondent respectivement aux topics et associations. Les topics $t_{1_{XTM2}}$ et $t_{2_{XTM2}}$ sont représentés par les *SP* désignés par t_1 et t_2. Les *SP* *comp*,*acq*, *ci*, *inst*, *class*, acq_{ing}, acq_{ed}, *cap*, *web*, *eng* et fr correspondent respectivement aux topics qui ont été créés pour représenter la classe `Company`, les types d'associations `Acquisition` et `Class-instance`, les types de rôles `instance`, `class`, `acquiring` et `acquired`, les types d'occurrences `Capital` et `WebSite` et les domaines de validité `English` et `Français`. Les *SP* "association" a_1, a_2 et a_3 représentent respectivement l'association d'acquisition et les deux associations de type `Class-instance`. Les *SP* "topic" t_1 et t_2 jouent respectivement les rôles acquéreur et acquis dans a_1 et le rôle de l'instance dans les *SP* "association" a_2 et a_3 au sein desquelles le *SP* "topic" *comp* joue le rôle de classe.

Après sa traduction en *SM*, l'élément TMDM correspondant au topic $t_{1_{XTM2}}$ devient le *SP* t_1 représenté à la figure 2.8. Il réunit dans les propriétés de clé *occurrences*, *topic names* et *roles played* ses noms, ses occurrences et les rôles qu'il joue dans les associations auxquelles il participe. La propriété de clé *subject identifiers* rassemble les identifiants des sujets représentés par ce topic. Les *SP* $r_{acq_{ing}}$ et r_{inst_1} correspondent respectivement aux rôles `Acquiring` et instance que le topic t_1 joue dans les associations de type `Acquisition` et `Class-instance`. Les *SP* occ_{web_1} et occ_{cap_1} correspondent aux occurrences de type `WebSite` et `Capital`. Lorsqu'un *SP* contient un autre *SP* dans une

de ses propriétés, le second possède une propriété de clé *parent* dont la valeur identifie le *SP* père. Par exemple, dans le cas de t_1 la valeur de cette propriété est tm puisque tm contient t_1 en tant que valeur d'une des propriétés de clé *topics*.

En ce qui concerne les associations, il en va presque de même que pour les topics. L'association a_1 représentant une acquisition de société est traduite en le *SP* de la figure 2.8. Comme cette association n'a pas de domaine de validité particulier, il lui en est attribué un par défaut. Le type de l'association est représenté par un *SP acq*. Chaque association de type `Class-instance` ou `Superclass-subclass` conduit respectivement à la création d'une relation isa_m ou sub_m établie entre les *SP* concernés. Le rôle $r_{acq_{ing}}$ présenté dans la même figure et apparaissant dans le *SP* de l'association a_1 se trouve être un des rôles joués par le topic t_1, puisqu'il apparaît en tant que valeur d'une propriété ayant pour clé $roles\,played$. Bien que t_2 ne soit pas évoqué ici, il joue un rôle $r_{acq_{ed}}$ de type `Acquired` dans l'association a_1, et un rôle de type instance dans une association a_3 représentant la relation de classe à instance qui le relie à un topic *comp* jouant le rôle de type classe.

Les opérations de base

Le langage des *SM* dispose de différentes primitives de navigation (présentées dans [DNB06]) :

- l'opérateur $\downarrow$ prend en argument un *SP* x à gauche et retourne l'ensemble des clés intervenant dans x, i.e. $x\downarrow = \{k|\exists v \in \mathcal{V} : (k,v) \in x\}$. Par exemple, $t_1\downarrow$ retourne les clés des propriétés du topic t_1, soit l'ensemble contenant : $topic\,names$, $occurrences$, $roles\,played$, $subject\,identifiers$ et $item\,identifiers$.
- l'opérateur $\uparrow$ prend en argument un *SP* x gauche et une *SM* m en indice. Il retourne l'ensemble des clés des *SP* y de m tel que x soit la valeur associée à chacune de ces clés, i.e. $x\uparrow_m = \{k|\exists y \in m : (k,x) \in y\}$. Par exemple, on considère l'ensemble T contenant les *SP* t, dont le libellé $lib(t)$ est la valeur d'une des propriétés de clé *topic* dans le *SP* tm. T étant un ensemble de *SP*, il est par définition une *SM*. Le résultat de $lib(sub_1)\uparrow_T$ est l'ensemble $\{subject\,identifiers\}$; la propriété $(lib(sub_1), subject\,identifiers)$ ayant été trouvée dans le *SP* t_1 identifié $lib(t_1)$ de la *SM* T.
- l'opérateur $\rightarrow$ prend en argument un *SP* x à gauche et un libellé k de $\mathcal{L}$ à droite. Il retourne l'ensemble des valeurs associées à la clé k dans le *SP* x, i.e. $x \rightarrow k = \{v|(k,v) \in x\}$. Son extension $x \rightarrow_m k*$ prend en plus une *SM* m en indice et retourne les valeurs associées à toutes les *sous-classes* de k, i.e $x \rightarrow_m k* = \{v|\exists k' \in \mathcal{L} : (k',v) \in x$ et $k' sub_m k\}$. Par exemple, $t_1 \rightarrow roles\,played$ retourne les rôles que joue le topic t_1 : soit l'ensemble $\{r_{acq_{ed}}, r_{inst_1}\}$.
- l'opérateur $\leftarrow$ prend en argument une valeur v de $\mathcal{V}$ à gauche, un libellé k de $\mathcal{L}$ à droite et une *SM* m en indice. Il retourne l'ensemble des *SP* de m qui contiennent

la propriété associant la clé k à la valeur passée en argument, i.e. $v \leftarrow_m k = \{x \in m | (k, v) \in x\}$. Son extension $v \leftarrow_m k*$ retourne les *SP* qui contiennent une propriété associant à une *sous-classe* de k la valeur passée en argument, i.e $v \leftarrow_m k* = \{x \in m | \exists k' \in \mathcal{L} : (k', v) \in x \text{ et } k' sub_m k\}$. Par exemple, sachant que $a_1 \rightarrow roles$ retourne les rôles joués dans l'association de types `Acquisition` sous la forme d'un ensemble de *SP* (et par conséquent une *SM*), $t_1 \leftarrow_{\{a_1 \rightarrow roles\}} player$ retourne l'ensemble des *SP* représentant le rôle que joue le topic t_1 dans l'association a_1 : soit $\{r_{acq_{ed}}\}$.

On généralise les trois premiers opérateurs pour qu'ils ne prennent en entrée non plus un seul *SP* mais un ensemble de *SP*. Le résultat de l'application d'un opérateur (un ensemble de *SP*) peut maintenant devenir l'argument d'un autre opérateur. En réitérant ce procédé, on créé des séquences (ou chemins) d'application de ces opérateurs qui peuvent être considérées comme les formules d'un langage nommé $\mathcal{P}_\mathcal{M}$. Ces formules forment un langage d'interrogation basique. Par exemple, on peut écrire une formule de $\mathcal{P}_\mathcal{M}$ qui demande à ce que soient retournées les sociétés acquises par la société représentée par t_1. En d'autres termes, elle retourne les topics connectés à t_1 par une association de type `Acquisition`, les premiers jouant le rôle d'acquis et le second le rôle d'acquéreur. Pour construire cette formule il faut procéder par étapes :

- On définit l'ensemble A des associations de tm avec $A = tm \rightarrow associations$,
- puis l'ensemble A_{acq} des associations de A de type `Acquisition` avec $A_{acq} = acq \leftarrow_A type$ (le *SP* correspondant à une association est donné en annexe),
- Ensuite on considère l'ensemble R_1 des rôles de chacune des associations de A_{acq} avec $R_1 = A_{acq} \rightarrow roles$,
- puis l'ensemble $R_{acq_{ing}}$ des rôles de R_1 de type `Acquiring` avec $R_{acq_{ing}} = acq_{ing} \leftarrow_{R_1} type$,
- et l'ensemble R_{t_1} des rôles de $R_{acq_{ing}}$ joués par t_1 avec $R_{t_1} = t_1 \leftarrow_{R_{acq_{ing}}} player$.
- On en déduit l'ensemble $A_{acq_{ing}}$ des associations de A_{acq} dans lesquelles t_1 joue le rôle `Acquiring` avec $A_{acq_{ing}} = R_{t_1} \rightarrow parent$
- On définit l'ensemble R_2 des rôles de chacune des associations de $A_{acq_{ing}}$ avec $R_2 = A_{acq_{ing}} \rightarrow roles$,
- puis l'ensemble $R_{acq_{ed}}$ des rôles de R_2 de type `Acquired` $R_{acq_{ed}} = acq_{ed} \leftarrow_{R_2} type$
- Et pour finir, le résultat, $T_{acq_{ed}}$ est l'ensemble des topics t de $R_{acq_{ed}}$ où t joue le rôle `Acquired`. Il est obtenu par $T_{acq_{ed}} = R_{acq_{ed}} \rightarrow player$. Pour l'exemple, $T_{acq_{ed}} = \{t_2\}$.

La formule entière s'obtient en remplaçant récursivement le nom de chaque ensemble par la définition qui lui est associée jusqu'à stabilisation de l'énoncé (c'est à dire jusqu'à ce que tout remplacement supplémentaire dans l'énoncé aboutisse à un énoncé identique), et ceci en considérant que l'énoncé de départ est $T_{acq_{ed}}$. La figure 2.9 présente la formule obtenue.

$$(acq_{ed} \leftarrow_{(t_1 \leftarrow_{acq_{ing}} \leftarrow_{(acq \leftarrow_{tm \rightarrow associations} type) \rightarrow roles} type player) \rightarrow parent) \rightarrow roles} type) \rightarrow player$$

FIG. 2.9 – La formule d'interrogation de SM permettant de calculer l'ensemble $T_{acq_{ed}}$ représentant les sociétés acquises par la société `MonitoringTech`.

En résumé, le formalisme des *SM* permet de modéliser des connaissances et d'y accéder en définissant des formules de parcours de chemins dans le réseau induit par ces connaissances. Les *SM* peuvent représenter des connaissances décrites en TM puisqu'une correspondance existe avec les objets *TMDM*. La suite présente un autre format qui établit une correspondance similaire.

2.3.3 Le langage Q.

Le format proposé dans [Gar05a] est un format élaboré à partir de quintuplets qui s'appelle le modèle Q et permet de représenter la structure *TMDM* correspondant à un document *XTM 2.0*. [Gar05b] présente un langage de requête pour les TM basé sur le modèle Q. Ce format est aussi utilisé pour représenter un document *RDF* et établir des transformations entre les TM et RDF.

Modèles de Q

On se donne un vocabulaire A partitionné en deux sous-ensembles I et L. I est l'ensemble des identifiants et L l'ensemble des littéraux (nombres ou chaînes de caractères). Un modèle de Q est un sous-ensemble de $I \times I \times I \times I \times A$. Soit $q = (a_0, a_1, a_2, a_3, a_4)$ un élément d'un de ces sous-ensembles. Les fonctions suivantes permettent d'accéder à l'élément a_i de q (avec i de 0 à 4 inclus).

- $subj(q) = a_0$ est le *sujet* sur lequel porte la mise en relation représentée par q.
- $pred(q) = a_1$ représente le *prédicat* qui met en relation deux éléments.
- $id(q) = a_2$ est *l'identifiant* du quintuplet.
- $con(q) = a_3$ représente le *contexte* de q sous la forme d'un identifiant. Il a été principalement introduit pour représenter en Q une TM (modélisée en TMDM).
- $val(q) = a_4$ est considéré comme étant *l'objet* de la mise en relation représentée par q.

Un modèle M est soumis aux contraintes suivantes :

- l'identifiant $id(q)$ de chaque $q \in M$ est unique au sein de M,
- un même quintuplet $q \in M$ ne peut apparaître deux fois avec des identifiants différents,
- l'identifiant $id(q)$ d'un quintuplet $q \in M$ ne peut être utilisé comme prédicat $pred(q)$ ni comme contexte $con(q)$

- le contexte $con(q)$ d'un quintuplet $q \in M$ ne peut être utilisé comme prédicat $pred(q)$

Représentation d'une TM par un modèle de Q

Un modèle de Q permet de représenter une instance de TMDM correspondant à un document *XTM 2.0*. La correspondance entre les deux formats n'est pas détaillée dans cette section. Un exemple est toutefois proposé en annexe à la section A.3.

D'une façon générale, un tel modèle reproduit fidèlement une partie de la structure de l'instance TMDM qu'il représente. Cette représentation s'appuie sur une partie immuable I_Q du vocabulaire I qui contient les termes utilisés en tant que prédicats de base pour définir des quintuplets de Q. Étant donné un quintuplet q, chaque $pred(q) \in I_Q$ donne à q une signification précise. Par exemple, lorsque $pred(q) = ScopeMember$, $subj(q)$ représente un domaine de validité composé du topic $obj(q)$. Ou encore, lorsque $pred(q) = Type$, $subj(q)$ est l'instance du type représenté par le topic $obj(q)$.

On remarque toutefois certaines différences structurelles mineures en ce qui concerne la représentation des associations. En effet, une représentation spécifique est réservée aux associations binaires. Une telle association est représentée par un quintuplet q où $id(q)$ est l'objet d'un quintuplet dont le prédicat est $Association$ et identifie l'association qui met en relation les topics sujet et objet respectivement identifiés par $subj(q)$ et $obj(q)$.

Lorsque l'arité de l'association est $n \neq 2$, le modèle contient n quintuplets q représentant le lien association-rôle-topic. Dans ce cas, $subj(q)$ est l'objet d'un autre quintuplet dont le prédicat est $Association$ et représente l'association en question dans laquelle $obj(q)$ représente un des topics parmi les n mis en relation, $id(q)$ identifie le rôle de type $pred(q)$ que joue le topic dans l'association. Les n quintuplets présentent un sujet identique.

Interrogation dans Q

Pour interroger les connaissances représentées dans le modèle Q, [Gar05b] propose un langage d'interrogation dont la syntaxe s'inspire du SQL ; le SQL étant le langage standard d'interrogation de systèmes de gestion de base de données (voir le document ISO/IEC de référence [90706]). En général, une requête de ce langage a pour forme :

SELECT x_1, x_2, ..., x_n

FROM $f(x_1, x_2, ..., x_n)$?

ou x_1, x_2, ..., x_n sont des variables et le point d'interrogation indique la fin de la requête. $f(x_1, x_2, ..., x_n)$ est une formule construite à partir de prédicats séparés par des connecteurs (AND, OR) ou pouvant faire l'objet d'une négation (NOT). Bien qu'une liste de prédicats prédéfinis soit disponible, le langage permet à l'utilisateur d'en créer de nouveaux en utilisant des opérateurs réservés à cet usage.

Pour de telles requêtes, le mécanisme d'interrogation retourne des ensembles de couples (x_i, v_i) qui assignent une valeur v_i à chaque variable x_i évoquée dans la tête de la requête à condition que cette assignation valide la formule $f(x_1, x_2, ..., x_n)$. Ce système d'interrogation est implémenté dans une application de gestion des connaissances [Gar05b].

La requête présentée à la figure 2.10 est définie pour rechercher les noms des sociétés acquises par la société MonitoringTechs. Plus précisément, on recherche le(s) nom(s) n_2 des topics t_2 qui joue(nt) un rôle $r_{acq_{ed}}$ de type "uri1 :Acquired" dans une association a de type "uri1 :Acquisition" au sein de laquelle le(s) topic(s) t_1 de nom "MonitoringTechs" joue(nt) un role $r_{acq_{ing}}$ de type "uri1 :Acquiring".

SELECT n_2
FROM
topic-name(t_1, "MonitoringTechs"),
role-player($r_{acq_{ing}}$, t_1), type($r_{acq_{ing}}$, "uri1 :Acquiring"),
association-role(a, $r_{acq_{ing}}$), association(a), type(a, "uri1 :Acquisition"),
association-role(a, $r_{acq_{ed}}$), type($r_{acq_{ed}}$, "uri1 :Acquired"),
role-player($r_{acq_{ed}}$, t_2),
topic-name(t_2, n_2) ?

FIG. 2.10 – Un exemple de requête permettant d'interroger une base en Q.

Les formats présentés jusqu'ici (voir 2.3.2 et 2.3.3) permettent de modéliser des TM qui respectent la syntaxe du *XTM 2.0* en établissant une correspondance avec les structures du format *TMDM*. Dans la suite, les formalismes présentés s'inspirent de la théorie des graphes pour représenter des TM exprimées en *XTM 1.0*.

2.3.4 Graphes Topic Maps

[BN01] propose d'interpréter formellement sous forme d'un graphe un document respectant la norme *XTM 1.0* et introduit le formalisme *TMG* (pour *Topic Maps Graphs*).

Les graphes Topic Maps (*TMG*)

Définition Un graphe Topic Map (*TMG*) est un tuple $(T, A, S, AM, label_{AM}, AS, AT, SC)$ avec T, A et S trois ensembles de sommets disjoints. T et S représentent respectivement les topics et les domaines de validité. Les éléments de A sont appelés "associations" mais

représentent à la fois les associations et les occurrences du format XTM. Pour éviter l'ambiguïté, on les nomme dans la suite *tmg*-association. AM, AS, AT et SC sont quatre ensembles d'arêtes disjoints qui respectent les conditions suivantes :

- **Pour les arêtes de type *AssociationMember* (AM)** : $AM \subseteq (A \times T) \cup A^2$ et les arêtes de AM sont toutes étiquetées par une fonction $label_{AM}$. Lorsque a représente une association (et non une occurrence) et t un topic, l'étiquette $l = label_{AM}(e)$ de l'arête $e = (a, x)$ de AM correspond au rôle de nom l que joue t dans a.
- **Pour les arêtes de type *AssociationScope* (AS)** : $AS \subseteq A \times S$ et $\forall a \in A, \exists (a, s) \in AS$. Chaque arête (a, s) de AS permet d'attribuer un domaine de validité s à une *tmg*-association a.
- **Pour les arêtes de type *AssociationTemplate* (AT)** : $AT \subseteq A \times T$. Soit (a, t) et (a', t') dans AT, si $t \neq t'$ alors $a \neq a'$. Une arête (a, t) de AT permet d'attribuer un type, représenté par le topic t, à l'association ou l'occurrence représentée par a.
- **Pour les arêtes de type *ScopeComponant* (SC)** : $SC \subseteq S \times T \cup S \times A$. Une arête (s, x) de SC permet de spécifier que le topic ou la *tmg*-association x participe à la définition du domaine de validité s.

L'entité représentée par un topic de TMG est appelée le *sujet*. Un sujet peut être (1) soit une ressource, (2) soit une entité dont la description est donnée par le contenu d'une ou plusieurs ressources. Dans le premier cas, le sujet est appelé *subject constituting resource* (S.C.R.) et dans le second *subject indicator resource* (S.I.R.[2]). On désigne par SCR et SIR les ensembles de *S.C.R.* et de *S.I.R* et on dispose d'une fonction *subjects* de $T \cup A$ dans $SCR \cup 2^{SIR}$ pour formaliser l'attribution d'une ressource à un topic en tant que *S.C.R.* ou de plusieurs ressources en tant que *S.I.R.*.

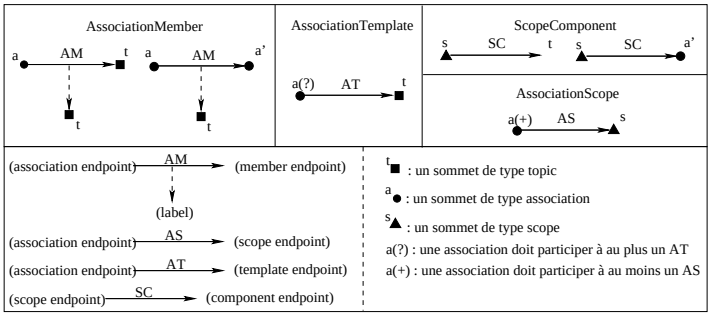


FIG. 2.11 – Les arêtes possibles dans un graphe Topic Map.

[2]Le *subject indicator resource* correspond à la notion de *source indicative d'information* introduite dans la partie 2.1.

Dans ce formalisme dit “graphique”, les “associations” représentent à la fois les occurrences et les associations du formalisme XTM. Les topics et les *tmg*-associations n'ont pas de type. L'étiquette associée par $label_{AM}$ à un arc de AM reliant une *tmg*-association a à un topic t correspondant à un nom de rôle lorsque a correspond à une association du format *XTM*. La figure 2.12 présente le graphe correspondant au document *XTM* regroupant les topics et l'association des figures 2.5 et 2.6. Les sommets t_1 et t_2 représentent les topics et a l'association d'acquisition qui les relie. La *tmg*-association *aci* représente quant à elle la relation entre instance (rôle *role-instance*) et classe (rôle *role-class*) établie par l'élément `<instanceOf>` dans un topic. Les topics t_1 et t_2 étant instances d'une même classe, ils sont reliés par cette *tmg*-association à un sommet topic représentant la classe `Company`. Ceci n'est valable que pour les topics et les occurrences. En effet, les *tmg*-associations sont reliées par un arc de AT (*associationTemplate*) à un topic considéré comme leur type. Dans cet exemple, le type de a est représenté par *acq*.

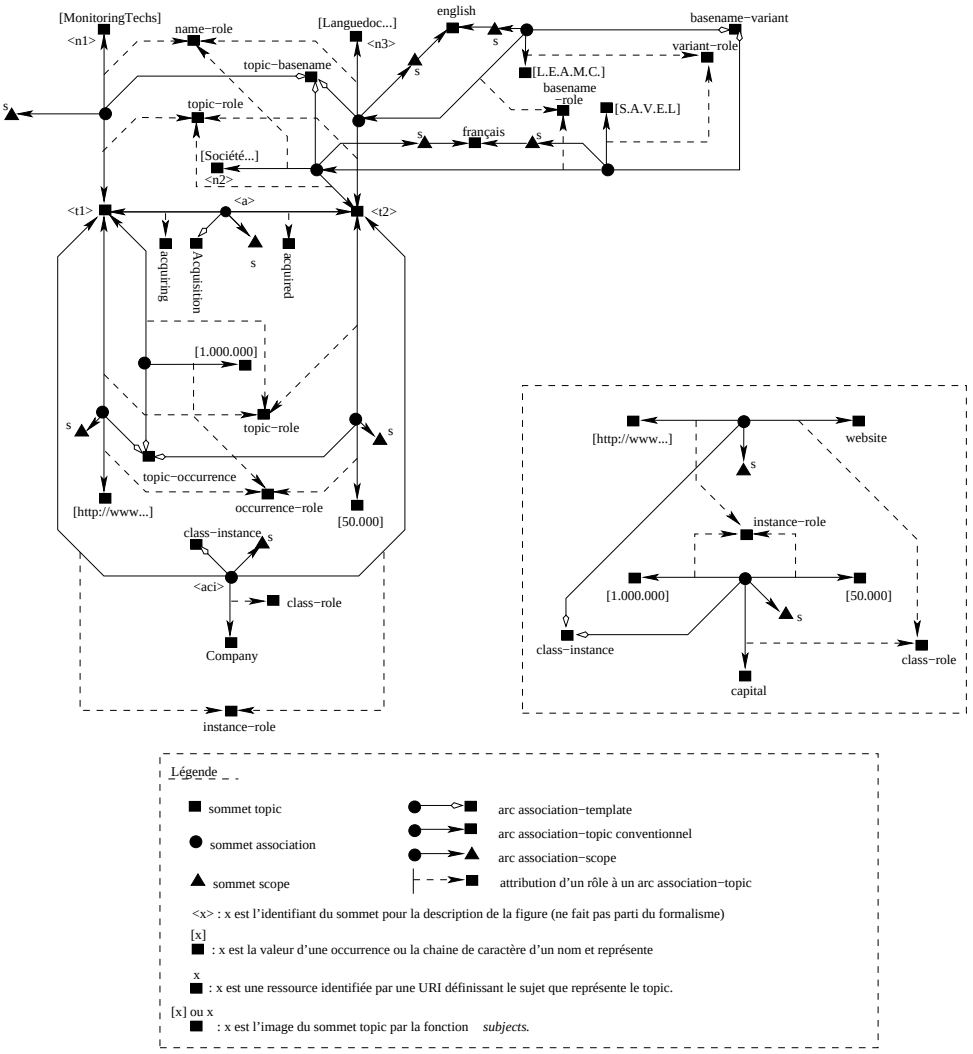


FIG. 2.12 – L'interprétation sous la forme d'une *TMG* du document *XTM* regroupant les topics et l'association des figures 2.5 et 2.6.

La fonction *subject* assigne à tous les topics les libellés qui les identifient dans le document XTM. Par exemple, *subject* assigne à t_1 le libellé "uri1 :Entité_Monitoring-Techs" dont *subjectIdentity* est parent. Les deux noms principaux de t_2 et celui de t_1 sont reliés par une *tmg*-association de type *topic-basename*. Un nom y joue le rôle *name-role* et un topic le rôle *topic-role*. Les libellés `[MonitoringTechs]`, `[Société...]` et `[Languedoc...]` désignent les images associées par *subjects* aux topics qu'ils étiquettent.

Ils correspondent aux libellés que `baseNameString` introduit dans les topics en question.

Dans le format *TMG*, toute *tmg*-association doit avoir un domaine de validité. Celui attribué à la *tmg*-association de type *topic-basename* dans laquelle le topic `[MonitoringTechs]` joue un rôle est le domaine de validité le moins contraint, c'est-à-dire vide. Celui attribué à la *tmg*-association dans laquelle `[Société...]` (resp. `[Languedoc...]`) joue un rôle est le sommet topic étiqueté *english* (resp. *français*) qui représente la langue anglaise (resp. française). Les libellés *english* et *français* représentent les URI dont *subjectIndicatorRef* est parent dans l'élément *scope* des noms de t_2. Chacune de ces URI est l'image par *subject* du topic qu'elle étiquette.

Les variations des noms `[Société...]` et `[Languedoc...]` sont représentées par les topics étiquetés `[S.A.V.E.L.]` et `[L.E.A.M.C.]`. Chaque variation est reliée par une *tmg*-association de type *basename-variant* à la *tmg*-association de type *topic-basename* dans lequel le nom est impliqué. La fonction *subject* permet d'attribuer à chaque variation de nom le libellé dont `variantName` est parent (i.e. `S.A.V.E.L.` et `L.E.A.M.C.`). Le domaine de validité d'un sommet étiqueté `[S.A.V.E.L.]` ou `[L.E.A.M.C.]` est celui du nom dont il est la variation.

En ce qui concerne les occurrences, l'occurrence de t_2 est représentée par un topic étiqueté `[50.000]` et celles de t_1 par deux topics étiquetés `[1.000.000]` et `[http://www...]`. Comme pour les noms et leurs variations, le libellé encodant la valeur d'occurrence introduite par `resourceData` est attribué par *subject* à chaque topic représentant une occurrence valeur. Chacune de ces trois occurrences est assignée à son topic respectif par une *tmg*-association de type *topic-occurrence* au sein de laquelle le topic représentant l'occurrence joue le rôle *occurrence-role* et celui représentant le topic porteur de l'occurrence joue le rôle *topic-role*. Un type est attribué à une occurrence de la même façon qu'à un topic par la *tmg*-association de type *class-instance*. Le sous-graphe représentant le typage des trois occurrences est représenté dans le rectangle en pointillé par mesure de lisibilité mais devrait être intégré avec l'autre graphe (en fusionnant les sommets de même nom). Les sommets étiquetés *website* et *capital* représentent les types d'occurrences.

Ce formalisme présente la particularité de permettre à deux *tmg*-associations d'être en relation par une arête de *AM*. On peut donc produire des *TMG* qui ne sont pas forcément expressibles en *XTM*.

En effet, une arête de *AM* représentant la mise en relation :

- d'une association *XTM* et d'un topic
- d'une occurrence *XTM* et d'un topic (les topics "portent" des occurrences)

est effectivement "naturellement" expressible en *XTM*. C'est à dire que ce langage a été conçu pour les exprimer aisément.

Cependant, lorsqu'une arête de *AM* représente un lien établi d'une association *XTM* :

- vers une autre association *XTM*

- ou vers une occurrence *XTM*

elle devient difficilement expressible en *XTM* : il faudrait établir une référence au sein de l'élément *XTM* `<association>` vers l'autre élément `<association>` ou `<occurrence>` en l'identifiant à l'aide de son identifiant *XML*. Si le lien s'établit vers une occurrence qui n'appartient à aucun sommet topic en TMG, cette dernière doit être introduite au sein d'un nouvel élément *XTM* `<topic>` supplémentaire (qui n'a aucun sommet correspondant en *TMG*) puisqu'elle ne peut exister seule à la racine d'un document *XTM*. Dans les deux cas, le document *XTM* est laborieux à définir et le résultat n'est pas très élégant.

Finalement, lorsqu'une arête de *AM* représente un lien établi d'une occurrence *XTM* :

- vers une association *XTM*
- ou vers une autre occurrence *XTM*

elle n'est pas expressible en *XTM*.

Bien que le formalisme *TMG* n'adopte pas précisément toutes les contraintes imposées par le format *XTM*, il permet de représenter une TM quelconque décrite dans ce format sous la forme d'un graphe (au sens théorie des graphes). Par contre la réciproque n'est pas vraie, certains *TMG* sont difficilement expressibles en *XTM*. Bien qu'aucune allusion n'y soit explicitement faite dans [BN01], le formalisme *TMG* se rapproche de la notion d'hypergraphe. En effet, les associations pourraient être considérées comme des multi-arêtes et les topics comme des sommets. La section suivante présente une approche qui s'inspire de cette idée.

2.3.5 Hyper-graphes Topic Maps

[AdMRV02] propose un formalisme basé sur la notion d'hypergraphe pour modéliser et structurer en différents niveaux sémantiques les connaissances exprimées en TM. Il est possible de procéder de deux manières différentes.

- La première approche consiste à représenter la connaissance d'une TM par deux hypergraphes que l'on notera $\mathcal{H}$ et $\mathcal{H}'$ et une fonction ϕ de l'ensemble des multi-arêtes de $\mathcal{H}'$ vers l'ensemble des sommets de $\mathcal{H}$. Le premier hypergraphe représente les éléments de la TM, c'est à dire les topics, les associations, les rôles, les noms et les occurrences. Alors que le second en association avec la fonction ϕ modélise le réseau induit dans la TM par la relation de typage et le domaine de validité des éléments. En d'autres termes, si dans une TM un topic est typé par un autre topic, une mise en relation correspondante est modélisée par $\mathcal{H}'$ et ϕ entre les sommets correspondant aux topics impliqués. Il en va de même pour l'attribution d'un domaine de validité.
- La seconde approche utilise l'hypergraphe $\mathcal{H}$ de la même façon que la première ; à ceci près qu'elle s'affranchit de l'hypergraphe $\mathcal{H}'$ et de la fonction ϕ. Une fonction qu'on appellera θ est utilisée pour le typage à la place de $\mathcal{H}'$ et de ϕ.

La notion d'hypergraphe

La définition d'hypergraphe employée dans [AdMRV02] est différente de celle - plus communément admise en théorie des graphes - n'impliquant qu'un ensemble de sommets et de multi-arêtes. En effet, les deux approches précédentes manipulent l'*incidence* d'une multi-arête sur un sommet en tant qu'objet mathématique. La définition d'hypergraphe doit donc tenir compte de cette notion d'incidence. Un hypergraphe est donc défini comme un quintuplet $\mathcal{H} = (V, E, I, \lambda_V, \lambda_E)$. V, E et I sont respectivement l'ensemble des sommets, des multi-arêtes et des incidences.

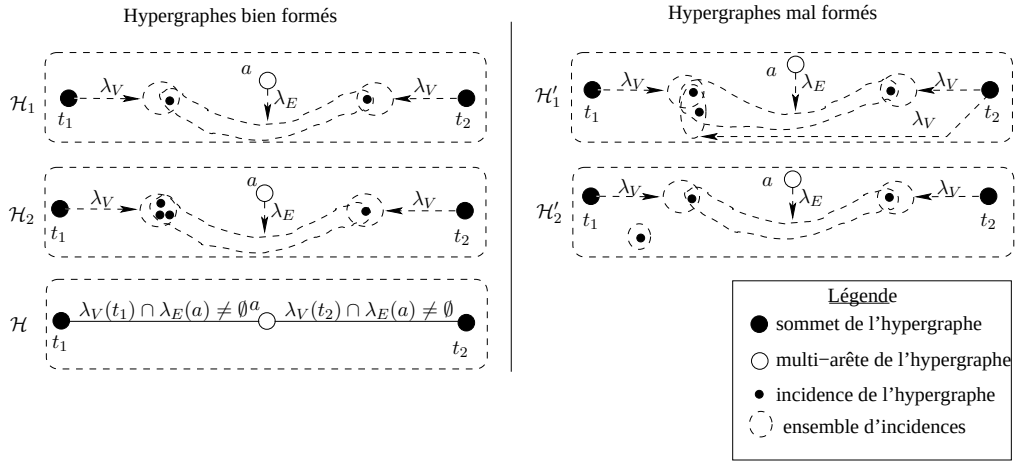


FIG. 2.13 – La représentation d'une association par un hypergraphe.

λ_V (resp. λ_E) est une fonction de V (resp. de E) dans $\mathcal{P}(I)$ telle que

- pour toute paire d'éléments différents de V (resp. de E) λ_V (resp. λ_E) associe à l'un un sous-ensemble de I disjoint de celui associé à l'autre ;
- l'union des ensembles appartenant au co-domaine de λ_V (resp. λ_E) est égale à I.

La figure 2.13 présente deux hypergraphes $\mathcal{H}_1$ et $\mathcal{H}_2$ bien formés et deux hypergraphes $\mathcal{H}'_1$ et $\mathcal{H}'_2$ mal formés qui représentent la mise en association par une multi-arête a de deux sommets t_1 et t_2. Les ensembles d'incidences images par λ_V dans $\mathcal{H}_1$ sont des singletons. Comme l'illustre la figure associée à $\mathcal{H}_2$ rien n'interdit à ces ensembles de contenir plusieurs incidences. Par contre, les hypergraphes $\mathcal{H}'_1$ et $\mathcal{H}'_2$ sont mal formés puisque le premier ne respecte pas la première condition énoncée sur λ_V et λ_E tandis que le second transgresse la seconde. Généralement, dans les figures, on représente les sommets de l'hypergraphe par des disques noirs et les multi-arêtes par des disques blancs. Si les ensembles d'incidences associés à un sommet t et à une multi-arête a ne sont pas disjoints, un segment relie les disques représentant t et a (comme le montre l'illustration

$\mathcal{H}$ de la figure 2.13).

Première approche

Pour représenter une TM par un hypergraphe, [AdMRV02] s'inspire de la correspondance établie dans [BN01] entre un graphe *TMG* et une TM. Les occurrences et les noms *XTM* sont donc considérés comme des associations. Une TM est représentée par un triplet $(\mathcal{H}, \mathcal{H}', \phi)$ où $\mathcal{H}$ et $\mathcal{H}'$ sont deux hypergraphes et ϕ est une fonction définie de l'ensemble des multi-arêtes de $\mathcal{H}'$ vers l'ensemble des sommets de $\mathcal{H}$.

L'hypergraphe $\mathcal{H} = (T, A, I, \lambda_T, \lambda_A)$ représente les éléments de la TM. Un topic est représenté par un sommet de T, une association par une multi-arête de A et le lien existant entre une association et un topic est représenté par un élément de I (il peut s'agir d'un rôle lorsque la multi-arête et le sommet considérés représentent respectivement une association et un topic du format *XTM*). Dans la figure 2.16, les topics t_1 et t_2 sont représentés par des sommets noirs et l'association qui les relie par un sommet blanc (une multi-arête de l'hypergraphe). Les incidences (segments entre l'association et les topics) correspondent aux rôles joués par ces topics dans cette association.

L'ensemble X des sommets de $\mathcal{H}' = (X, M, I', \lambda_X, \lambda_M)$ contient les sommets, les associations et les incidences de l'hypergraphe $\mathcal{H}$. $\mathcal{H}'$ permet de regrouper en plusieurs "classes" certains éléments de $X = T \cup A \cup I$ en les reliant par des multi-arêtes de M appelées *méta-associations*. Les multi-incidences dans $\mathcal{H}'$ ne sont pas permises ; c'est à dire que pour tout sommet x de X et toute multi-arête m de M l'ensemble $\lambda_X(x) \cap \lambda_M(m)$ est réduit à au plus un élément. L'hypergraphe $\mathcal{H}_1$ de la figure 2.13 vérifie cette condition : les ensembles $\lambda_V(t_1) \cap \lambda_E(a)$ et $\lambda_V(t_2) \cap \lambda_E(a)$ sont des singletons.

La fonction ϕ est injective et associe à chaque méta-association m de M un sommet topic de $\mathcal{H}$. ϕ ne doit pas associer à m un sommet topic de T si ce dernier appartient à une composante connexe de $\mathcal{H}$ contenant un des sommets reliés par m (contrainte de *non-connexité*).

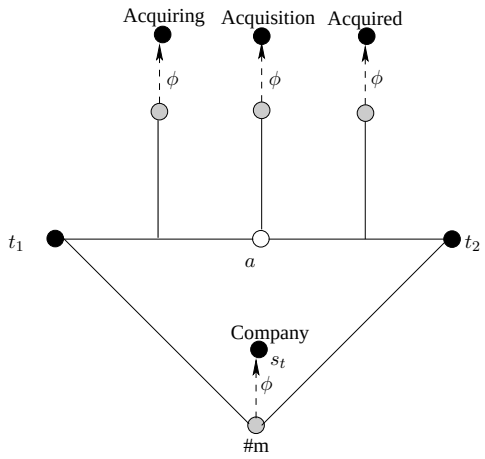


FIG. 2.14 – L'attribution d'un type par ϕ aux sommets t_1 et t_2, à l'association a et aux rôles.

La notion de "typage" en TM est modélisée par l'hypergraphe $\mathcal{H}'$ et la fonction ϕ. Par exemple, on considère le document *XTM* où deux topics sont d'un même type `Company` lui-même étant représenté par un troisième topic. Comme le montre la figure 2.14, on associe à ces deux topics deux sommets t_1 et t_2 différents, tous deux appartenant à $\mathcal{H}$ et $\mathcal{H}'$. Puis, on créé dans $\mathcal{H}'$ une méta-association m qui relie t_1 et t_2. On associe ensuite au topic représentant le type `Company` un sommet s_t dans $\mathcal{H}$ et on précise que $\phi(m) = s_t$. Cela revient à dire que tous les éléments de la TM correspondant aux sommets reliés par m sont de type t. La contrainte de *non-connexité* garantit qu'il n'existe pas de chemin entre une instance et le type qui lui est associé par ϕ et $\mathcal{H}'$. Par exemple, on est assuré qu'à partir des topics t_1 ou t_2 il n'y a pas de chemin formé d'associations menant au topic *Company* (ce qui constituerait un non-sens). On procède de la même manière pour les rôles et les associations. L'attribution d'un domaine de validité à une association, une occurrence ou un nom est aussi modélisée en utilisant la fonction ϕ. Dans la figure 2.14, les sommets, les associations et les incidences de $\mathcal{H}$ sont respectivement représentés par des disques noirs, des disques blancs et par des segments entre deux disques de couleurs différentes. Les méta-associations de $\mathcal{H}'$ sont représentées par des disques gris connectés aux sommets, associations et incidences de $\mathcal{H}$.

Seconde approche

Une alternative équivalente à l'approche précédente consiste à ne modéliser une TM que par un seul hypergraphe noté $\mathcal{H}$ et une fonction θ. Comme précédemment, l'hypergraphe $\mathcal{H}$ permet de représenter les éléments de la TM et $\theta : T \rightarrow 2^{T \cup A \cup I}$ permet

d'associer à un sommet topic t de T un sous-ensemble E de $T \cup A \cup I$. θ doit respecter la même contrainte que ϕ à savoir que l'image par θ d'un topic t doit toujours être disjointe de la composante connexe à laquelle il appartient ; et ceci implique que θ n'a aucun point fixe (i.e. aucun $x \in T$ tel que $x \in \theta(x)$). La figure 2.15 illustre une utilisation de la fonction θ pour typer les sommets, associations et incidences de l'hypergraphe $\mathcal{H}$ présenté à la figure 2.14.

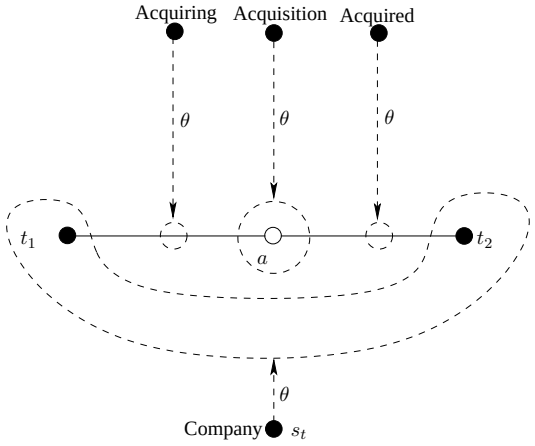


FIG. 2.15 – L'attribution d'un type par θ aux sommets t_1 et t_2, à l'association a et aux rôles.

En *TM*, $\theta(t) = E$ signifie que les topics ou associations (resp. rôles) correspondant aux élément de E sont *typés* (resp *regroupés* sous un même nom) par le topic correspondant au sommet t. Dans la figure 2.16, l'attribution d'un sous-ensemble $\theta(t)$ de $T \cup A \cup I$ en tant qu'image par θ d'un élément t de T est représentée par des flèches en pointillés. L'ensemble $\theta(t)$ est constitué des éléments pointés par les flèches partant de t. Par exemple, comme t_1 appartient à l'image par θ du sommet `Company`, on considère qu'il est de type `Company`. De la même façon que pour ϕ, l'attribution d'un domaine de validité à une association, une occurrence ou un nom est aussi modélisée par θ.

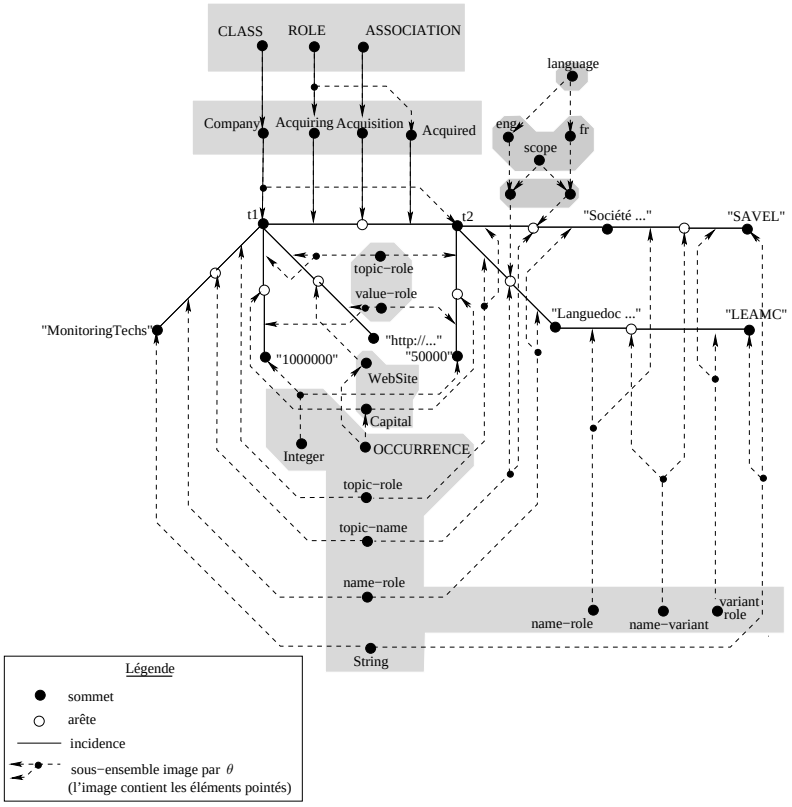


FIG. 2.16 – La représentation en hypergraphe de la TM composée des topics et de l'associations présentés en figures 2.5 et 2.6.

Il est parfois nécessaire, par mesure de rigueur, d'imposer à tout topic d'une TM d'être instance d'un topic. Comme une TM comporte un nombre fini de topics, cela signifie qu'un cycle existe dans la relation de typage. Dans certains cas, ce cycle est une boucle : par exemple, si on souhaite modéliser le fait que le topic *classe* est instance de lui même. Comme la fonction θ n'admet aucun point fixe x du type $x \in \theta(x)$ et que, dans ce cas ci, le topic appartient à la même composante connexe que son instance, il est impossible d'utiliser θ dans ce cas de figure. La fonction θ peut néanmoins former des cycles autres que des boucles. En effet, l'hypergraphe $\mathcal{H}$ associé à une TM peut contenir n composantes connexes C_i chacune vérifiant soit qu'il existe $t_i \in C_i$ tel que $\theta(t_i) \cap C_{i+1} \neq \emptyset$ pour i de 0 à $n-1$; soit que $\theta(t_n) \cap C_0 \neq \emptyset$.

En pratique, ce type de TM est peu utilisé. La préférence porte sur les TM dont l'hypergraphe associé ne forme aucun cycle avec la fonction θ. On dit de ces TM qu'elles sont *stratifiées*. Dans une TM stratifiée on peut partiellement ordonner l'ensemble des

composantes connexes de telle sorte que si une composante connexe C contient les éléments associés par θ à un topic d'une autre composante C', alors on écrit $C <_\theta C'$. L'ordre partiel induit par θ permet d'organiser la connaissance modélisée dans une TM en différents niveaux sémantiques. La figure 2.16 représente une TM stratifiée. La composante inférieure à toutes les autres dans l'ordre $<_\theta$ correspond à celle contenant les éléments qui ne sont l'antécédent d'aucun autre par θ. Dans la figure, elle ne compte aucun sommet de la portion grisée et contient le sommet t_1. La composante supérieure à toutes les autres est celle dont les sommets ne sont l'image d'aucun autre par θ. Dans la figure, elle est coloriée en gris et contient les sommets *Class*, *Occurrence* et *value-rôle*. Celle au niveau intermédiaire contient des éléments qui sont à la fois l'image et l'antécédent par θ d'autres éléments. Elle est de couleur grise et les sommets *Acquisition* et *Capital* en font partie.

Cette représentation basée sur les hypergraphes a conduit à une première méthode pour hiérarchiser les connaissances représentées dans une TM. L'approche présentée dans la suite s'inspire encore de la théorie des graphes et propose aussi une organisation par niveaux des connaissances d'une TM. Elle est basée sur la notion de graphe biparti et a été définie en fonction des besoins du logiciel ITM développé par Mondeca.

2.4 Les Topic Maps utilisées à Mondeca.

Le langage TM utilisé à Mondeca s'inspire de très près du langage standard *1.0* des TM mais comporte quelques différences. La *dtd* de la syntaxe *XTM* utilisée est donnée en annexe. Dans la suite, on notera XTM_{MDK} le format *XTM* utilisé à Mondeca. Cette partie présente une syntaxe abstraite - que l'on nommera $XTMG_{MDK}$ - pour le langage XTM_{MDK}.

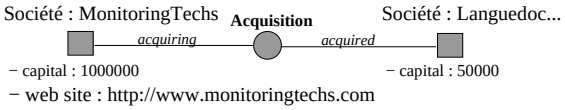


FIG. 2.17 – Une TM.

Une TM en $XTMG_{MDK}$ est un graphe biparti étiqueté $tm = (T, A, E, type, nom)$ composé de sommets topics T et de sommets associations A. Les arêtes $E \subseteq A \times T$ indiquent la participation des topics aux associations. *type* est une fonction d'étiquetage de $T \cup A \cup E$ dans L_T où L_T est un ensemble de libellés identifiant des topics. La fonction *type* associe à chaque sommet et arête un libellé de L_T spécifiant selon le cas la classe (d'un topic), le type (d'une association) et le rôle (d'un lien de participation). La fonction partielle *nom* de T dans L_T permet d'identifier certains topics par un libellé.

On peut compléter la description d'un topic et d'une association par un ensemble d'occurrences qui indiquent une information pertinente pour le topic ou l'association. Chaque occurrence est composée d'une valeur v, d'un type de donnée d (appelé son *type physique* dans ITM), et du nom l du topic qui type l'occurrence (appelé *type logique* dans ITM). On dispose donc d'une fonction *occ* de $T \cup A$ dans $2^{V \times D \times L_T}$ qui associe à un topic ou une association l'ensemble de ses occurrences, où D est l'ensemble des types de données, et V l'ensemble des valeurs des types de D. L'ensemble V contient les libellés de L_T et un élément particulier noté $\varnothing$. Un triplet (v, d, t) appartenant à l'ensemble associé à un topic par *occ* représente une occurrence dont le type de donnée est d, le type logique est t et la valeur est v (valeur dite *indéfinie* lorsque $v = \varnothing$).

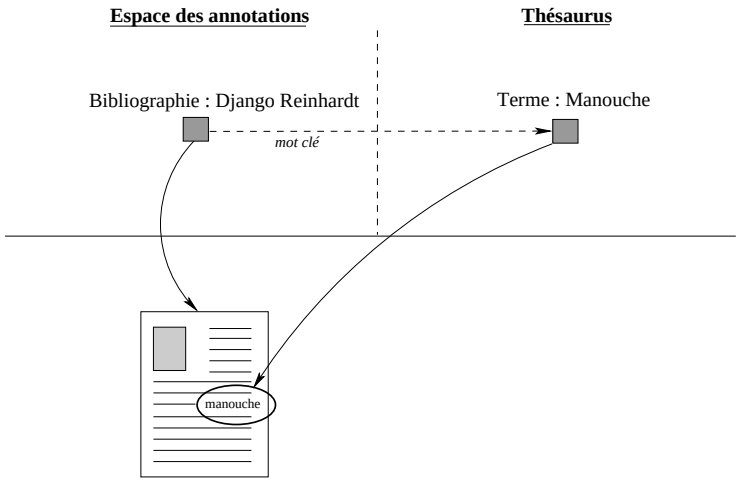


FIG. 2.18 – Un topic qui "pointe" vers un autre topic.

La TM de la figure 2.17 associe au topic de nom `MonitoringTechs` deux occurrences : la valeur `1000000` de type physique `Entier` et type logique `capital`, et la valeur `http://www.monitoringtechs.com` de type physique `URL` et type logique `web site`. Ces occurrences peuvent s'interpréter comme le capital de la société Oracle et les adresses web permettant de la contacter.

ITM permet la création d'occurrences particulières de type physique `pointeur` pour établir des relations "directes" entre topics (i.e. sans créer d'association) ou entre une association et un topic. On peut ainsi relier des topics d'espaces différents (par exemple, associer un terme du thesaurus à une ressource de l'espace d'annotation). En $XTMG_{MDK}$, une telle occurrence se représente par un triplet $(l_{t_2}, d, l_{t_{pointeur}})$ appartenant à l'ensemble associé par *occ* à un topic t_1 ; $l_{t_{pointeur}}$ étant le libellé identifiant le topic représentant le type physique `Pointeur`. l_{t_2} est un libellé identifiant dans ITM le topic pointé, c'est-à-

dire qu'il existe dans ITM une TM $tm = (T, A, E, type, nom)$ et un topic $t_2 \in T$ tel que $l_{t_2} = nom(t_2)$. La figure 2.18 illustre un topic possédant un pointeur vers un autre topic.

Il faut remarquer qu'ITM n'exploite pas toujours entièrement le formalisme *XTM*, en modifie son interprétation et le déforme parfois un peu. Tout d'abord, ITM permet d'attribuer des occurrences aux associations, alors que cela n'est pas permis en *XTM*. De plus, l'usage qu'ITM réserve aux domaines de validité est relativement pauvre puisqu'ils ne sont utilisés que pour attribuer une langue à un nom de topic. Enfin, l'utilisation des noms de topic dans ITM n'est pas toujours conforme aux contraintes imposées par le formalisme *XTM*. En effet, il n'est pas interdit à un topic de porter plusieurs noms, ni à deux topics qui représentent des entités différentes de porter des noms identiques. Le topic n'est pas identifié par son nom mais par son PSI. Pour tenir compte de cela, la fonction *nom* attribue à un topic l'URI de son PSI et non pas le nom de ce topic. Techniquement, le nom d'un topic est représenté dans ITM par une occurrence $(l_t, String, TopicName)$ où l_t est le nom du topic, $TopicName$ est le type logique de l'occurrence réservé aux noms de topic et $String$ le type physique "chaîne de caractères" de l'occurrence. Par exemple, pour le topic $t_{1_{XTM}}$ (présenté en figure 2.5) de nom "`MonitoringTechs`", la fonction *nom* retourne le libellé `uri1 :Entité_MonitoringTechs` et le triplet $(MonitoringTechs, String, TopicName)$ apparaît dans l'ensemble retourné par $occ(t_{1_{XTM}})$.

Dans la suite, on présente quelques définitions et propriétés remarquables des TM.

TM complète et fonctions de référencement

On se munit d'une fonction Ref qui associe à une TM le sous-ensemble des libellés de L_T auxquels font référence ses fonctions *type* et *occ*, et on définit une fonction Nom, l'extension à une TM de la fonction *nom*, qui associe à une TM le sous-ensemble des libellés de L_T images d'un topic par *nom*.

Une TM tm est dite *complète* si tout libellé utilisé par les fonctions *type* et *occ* est associé à un topic par la fonction *nom* : $Ref(tm) \subseteq Nom(tm)$.

On définit aussi la fonction $RefType$ qui associe à une TM tm le sous-ensemble des libellés de $Ref(tm)$ qui désignent un type (d'un topic, d'une association, d'un rôle ou d'une occurrence).

Éléments et inclusion de TM

On dispose d'une fonction *elt* qui retourne les composants d'une TM, dits *les éléments* d'une TM. *elt* prend en argument une TM $tm = (T, A, E, occ, nom, type)$ et retourne l'ensemble des *éléments* de tm telle que $elt(tm) = T \cup A \cup E \cup OTA \cup NT \cup Types$ avec :

- $OTA = \{\ (o, x) \in (V \times D \times L_T) \times (T \cup A)$ tel que $x \in T \cup A$ et $o \in occ(x)\}$
- $NT = \{\ (n, t) \in L_T \times T$ tel que $t \in T$ et $n = nom(t)\ \}$

– $Types = \{ (\tau, x) \in L_T \times (T \cup A \cup E)$ tel que $\tau = type(x) \}$

L'inclusion de tm dans tm' s'écrit alors $tm \subseteq_{TM} tm'$ et est vérifiée ssi $elt(tm) \subseteq elt(tm')$. On dit aussi que tm' contient tm.

Une TM tm est *la plus grande TM incluse* dans une autre TM tm' dont chaque élément $x \in elt(tm)$ vérifie indépendamment des autres une certaine propriété ssi tm contient toute TM incluse dans tm' dont les éléments vérifient la même propriété. On est assuré du caractère unique de tm.

Isomorphisme étiqueté de TM

On définit une relation d'isomorphisme étiqueté adaptée aux TM et établie par la fonction iso_{TM}. Soit deux TM $tm = (T, A, E, occ, nom, type)$ et $tm' = (T', A', E', occ', nom', type')$, iso_{TM} est une fonction qui associe à tout élément de $T \cup A$ un élément de $T' \cup A'$. iso_{TM} définit un isomorphisme étiqueté entre les TM tm et tm' lorsque :

– d'une part $e = (x, a) \in E$ ssi $e' = (iso_{TM}(x), iso_{TM}(a)) \in E'$ et $type(e) = type(e')$,
– d'autre part, pour tout $x \in T \cup A$ $occ(x) = occ(iso_{TM}(x))$, $type(x) = type(iso_{TM}(x))$,
– et enfin, pour tout $nom(x) = nom'(iso_{TM}(x))$.

Dans le cas où les fonctions nom, occ et $type$ ne sont pas définies pour être appliquées sur un élément x, il doit en être de même pour l'application des fonctions nom', $type'$ et occ' sur l'élément $iso_{TM}(x)$ (et réciproquement).

2.4.1 Extension de la formalisation

Le formalisme $XTMG_{MDK}$ proposé pour le langage XTM_{MDK} ne couvre pas les notions de domaine de validité, de variation de nom et d'identification de sujet car ces notions n'étaient pas nécessaires pour la réalisation des travaux de thèse. Toutefois, il peut être utile d'en tenir compte à l'avenir. Cette section propose des pistes pour élargir la définition d'une TM donnée en partie 2.4 en la complétant de trois nouvelles fonctions *subjects*, *var* et *scope*. Dans cette partie la fonction *nom* s'aligne sur la notion de *nom de topic* développée par le standard *XTM* (et non pas sur celle particulière à ITM). En d'autres termes, le libellé de L_T éventuellement attribué par *nom* à un topic correspond au *nom* que ce topic porte dans le document *XTM* (et non pas à son PSI).

Dans le standard *XTM 1.0*, il est possible d'attribuer un domaine de validité aux noms, aux occurrences et aux associations. L'attribution d'un domaine de validité est représentée en $XTMG_{MDK}$ par les fonctions $scope_N$, $scope_O$ et $scope_A$. Les fonctions $scope_A$ et $scope_N$ assignent respectivement à toute association a et tout couple $(t, nom(t))$ un sous-ensemble éventuellement vide de topics de T. La fonction $scope_O$ assigne à tout couple (t, o) avec $o \in occ(t)$ un sous-ensemble de T. $scope_N$, $scope_O$ et $scope_A$ sont donc des fonctions partielles ayant 2^T pour co-domaine et respectivement $T \times L_T$, $T \times O$ et A pour domaine. Pour simplifier l'usage de ces trois fonctions, une fonction *scope* est

définie de l'union des domaines de chacune des fonctions précédentes vers l'union de leurs co-domaines. L'utilisation de *scope* équivaut à appliquer la fonction parmi les trois précédentes qui a été définie pour prendre en argument celui passé à *scope*.

La figure 2.19 illustre l'attribution d'un domaine de validité aux noms du topic $t_{2_{XTM}}$ présenté en figure 2.5 et modélisé par $scope(t_{2_{XTM}}, "Societe...") = \{fr\}$ et $scope(t_{2_{XTM}}, "Languedoc...") = \{eng\}$.

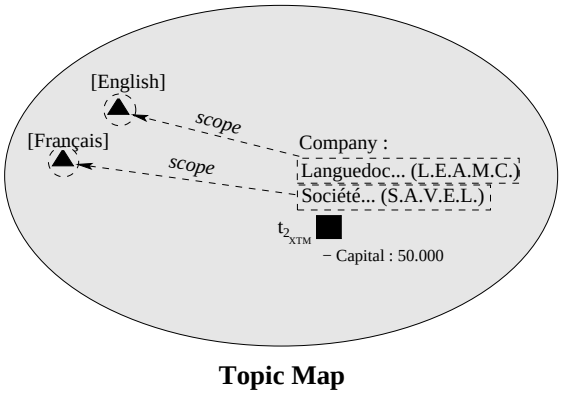


FIG. 2.19 – Un exemple d'application de la fonction *scope*.

On peut modéliser les variations de noms par une fonction partielle notée *var* de domaine $T \times L_T$ et de codomaine L_T qui à un couple (t, l_t) associe un libellé l'_t de L_T à condition que l_t soit le nom du topic t. *var* ne permet pas de modéliser l'attribution récursive d'une variation à une autre ; cette possibilité étant d'une utilité discutable puisqu'elle a été abandonnée par le format *XTM 2.0* et n'est pas exploitée dans ITM.

Le domaine de validité d'une variation de nom de topic est le domaine de validité du nom du topic : c'est à dire que $scope(t, var(t, l_t)) = scope(t, l_t)$, l_t étant le nom $nom(t)$ du topic t. Dans l'exemple, les variations des noms du topic $t_{2_{XTM}}$ sont modélisées par $var(t_{2_{XTM}}, "Societe...") = "S.A.V.E.L."$ et $var(t_{2_{XTM}}, "Languedoc...") = "L.E.A.M.C."$.

Le standard XTM *1.0* permet d'identifier une ressource qui *est* le sujet représenté par le topic. Pour représenter cela en $XTMG_{MDK}$, on introduit une fonction *subject* et un ensemble R de ressources tels que la fonction associe à un topic t de T *la* ressource r de R que t représente. Dans le cas où le sujet n'est pas une ressource identifiable, le standard permet à un topic de désigner des sources indicatives d'informations qui *définissent* le sujet représenté par le topic t (par exemple, une page web qui présente une définition). Dans ce cas, la fonction *subject* associe à t un sous-ensemble de R.

Si on considère un topic $t \in T$ et une ressource $r \in R$, $subject(t) = r$ signifie que

la ressource r est représentée par t. Par contre, $subject(t) = \{r\}$ signifie que ce qui est décrit (ou défini) par r est représenté par t. Enfin, le standard permet à un topic t de représenter un autre topic t' de la TM courante : on dit qu'il réifie ce topic. Pour modéliser cette situation on peut étendre le codomaine de *subject*, afin qu'elle puisse associer à un topic t un autre topic t'. La fonction *subject* est donc définie avec un domaine T et un codomaine $R \cup 2^R \cup T$.

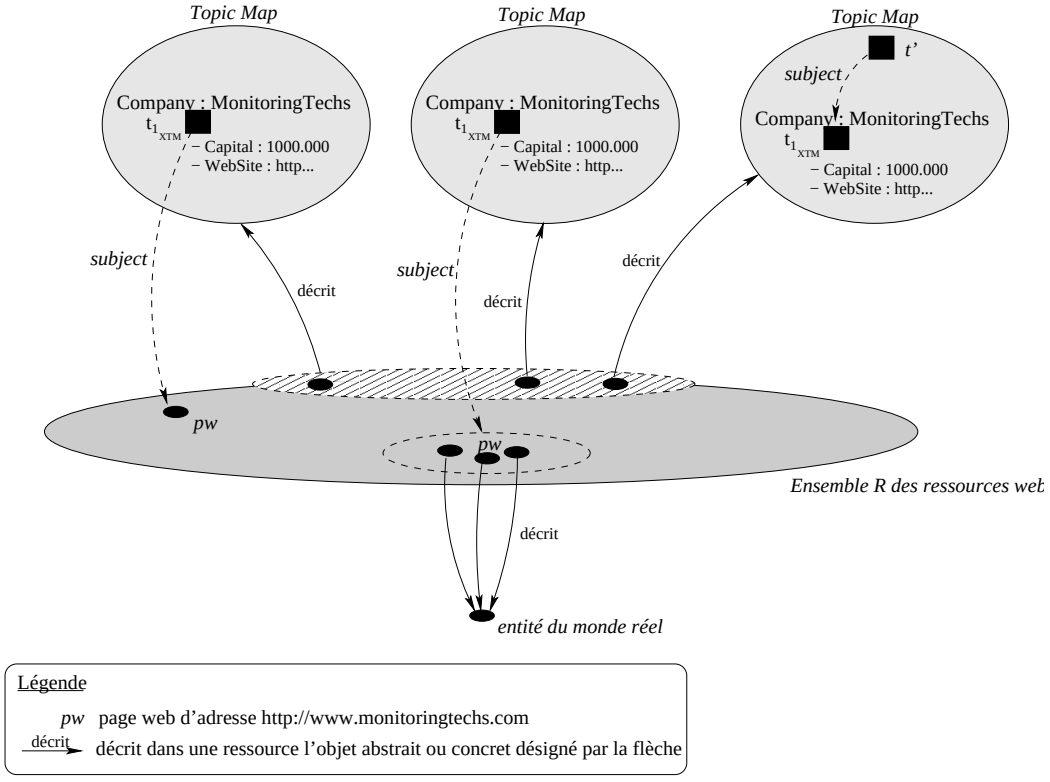


FIG. 2.20 – Les trois cas possibles d'application de la fonction *subject*.

La figure 2.20 présente les trois cas concernant *subject*. Dans la TM de gauche, $t_{1_{XTM}}$ représente la page web *pw* identifiée par `http://`www.monitoringtechs.com. Ce cas est modélisé par $subject(t_{1_{XTM}}) = pw$. Dans la TM du centre, $t_{1_{XTM}}$ désigne la page web *pw* comme l'une des sources indicatives d'informations qui caractérisent l'entité représentée. Dans ce cas, $subject(t_{1_{XTM}})$ désigne un ensemble et *pw* doit appartenir à R et à $subject(t_{1_{XTM}})$. Dans la dernière TM, le topic $t_{1_{XTM}}$ est réifié par le topic t'. Cela revient à dire que t' représente $t_{1_{XTM}}$ et ce cas est modélisé par $subject(t') = t_{1_{XTM}}$.

La définition formelle du quintuplet $tm = (T, A, E, type, nom)$ correspondant à une

TM d'ITM devient donc une structure $tm = (T, A, E, R, type, nom, var, scope, subject)$ avec T l'ensemble des sommets topics, A l'ensemble des sommets associations, E les arêtes qui indiquent le rôle joué par un topic lorsqu'il participe à une association, R un ensemble de ressources suivi des fonctions qui viennent d'être présentées. Il faut toutefois noter que cette extension du modèle $XTMG_{MDK}$ n'est pas exploitée dans la suite de ce travail. Elle a été développée pour pouvoir facilement prendre en compte les nouveaux besoins qui pourraient apparaître en matière de représentation ; le secteur industriel étant soumis à des évolutions constantes.

2.4.2 Les trois niveaux de connaissances.

Une base de connaissances ITM forme une TM structurée en trois niveaux par l'association *classe-instance* (un exemple est donné en figure 2.22) ; chacun des niveaux formant une TM à part entière.

- la TM tm_{meta} du niveau méta décrit le méta-modèle réflexif des connaissances d'ITM (cf. cadre `Niveau Méta` de la figure 2.22) ;
- la TM tm_{modele} du niveau modèle est définie en utilisant les primitives du niveau méta. Ce niveau est généralement composé d'une ontologie de domaine spécifiant la sémantique du vocabulaire utilisé pour décrire (au niveau sémantique) le contenu des informations gérées par la base, et de plusieurs ontologies de "services" spécifiant la sémantique des représentations dédiées à la mise en œuvre de services spécifiques d'ITM (primitives de représentation de thesaurus, primitives de représentation de l'organisation logique des documents...) ;
- enfin la TM $tm_{instances}$ au niveau instances contient les instances de la base qui sont "contrôlées" par le modèle du niveau supérieur. On y trouve des annotations sémantiques qui décrivent le contenu des informations gérées par la base, des ressources terminologiques (thesaurus), la description de l'organisation logique des documents...

La TM de niveau méta est construite à partir d'un vocabulaire qui varie très peu. Ce vocabulaire est interprété par ITM pour doter les bases de connaissances de la sémantique opérationnelle suivante[3].

[3]Toutes les opérations effectuées dans ITM s'appuient sur cette sémantique opérationnelle.

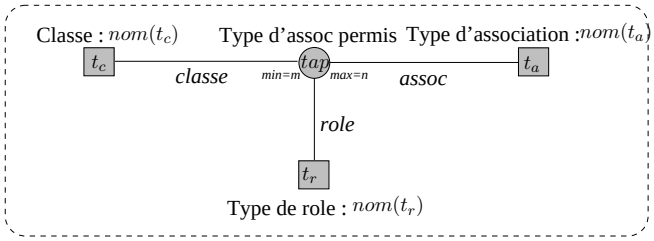


FIG. 2.21 – Le schéma type d'une définition utilisant la primitive `type d'association permis`.

Une association *tap* de type `type d'association permis` reliant trois topics t_c, t_a et t_r spécifie qu'il est permis au niveau inférieur à un topic de classe t_c de jouer le rôle t_r dans l'association de type t_a. Si on spécifie une occurrence ayant comme valeur un entier n de type logique `cardinalité maximum` (resp. `cardinalité minimum`) sur l'association *tap*, cela signifie qu'une association de type t_a doit admettre au maximum (resp. minimum) n rôle(s) de type t_r. La figure 2.21 présente le schéma type d'une définition au niveau modèle utilisant une primitive `type d'association permis`.

Une association *top* de type `type d'occurrence permis` reliant t_c et t_o permet à un topic de classe t_c de posséder une occurrence de type logique t_o. On peut spécifier sur *top* une occurrence de type `cardinalité maximum` (resp. `cardinalité minimum`) et de valeur n qui signifie qu'un topic de classe t_c doit comporter au maximum (resp. minimum) n occurrence(s) de type t_o. La figure 2.23 présente le schéma type d'une définition avec la primitive `type d'occurrence permis`.

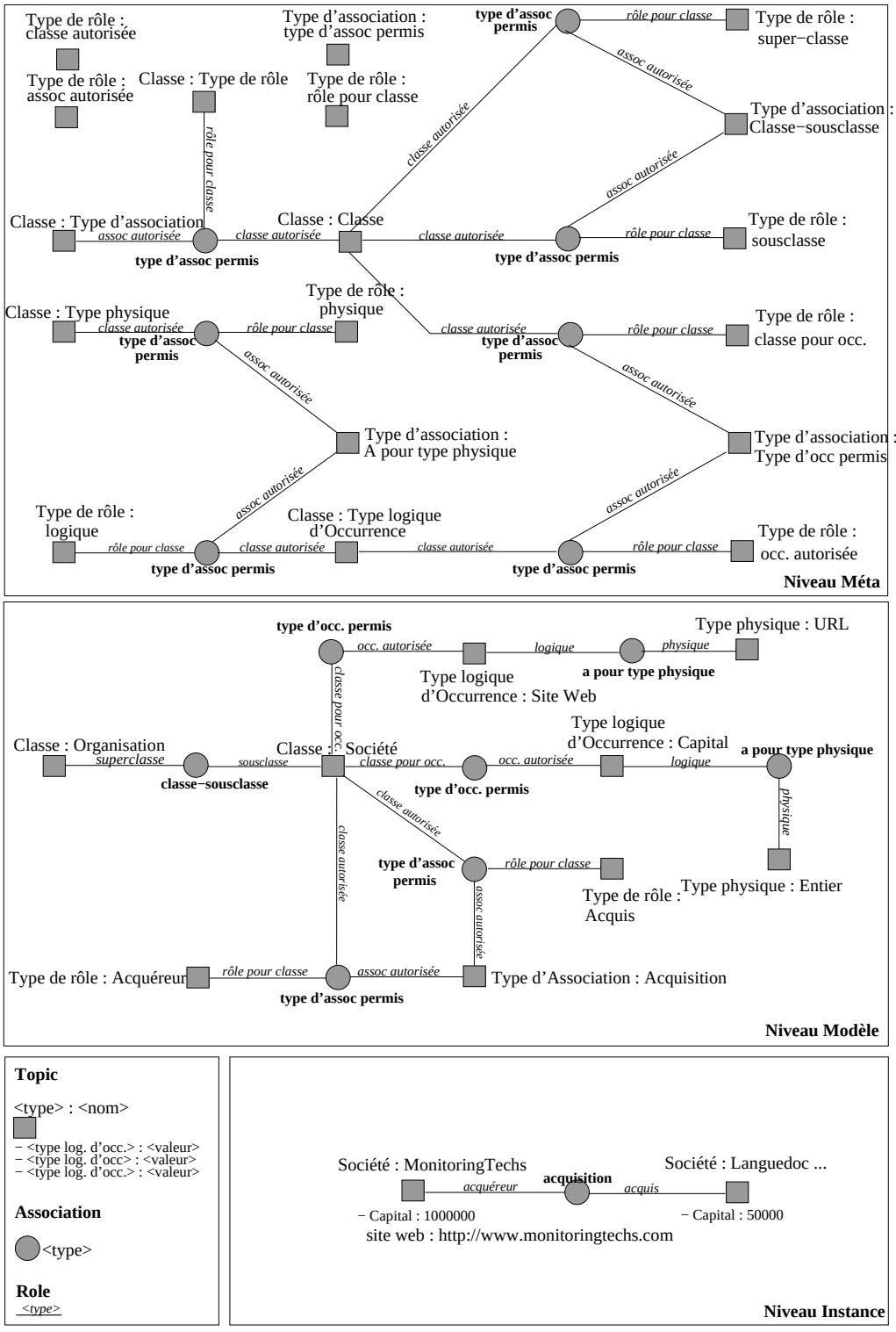


FIG. 2.22 – Un extrait de base ITM avec son niveau méta, modèle et instance.

Une association *ptp* de type `a pour type physique` reliant t_o et t_p spécifie que toute occurrence de type logique t_o a pour type physique t_p ce qui permet de contraindre la valeur de l'occurrence (donnée numérique, textuelle, format date, pointeur) et de l'interpréter correctement.

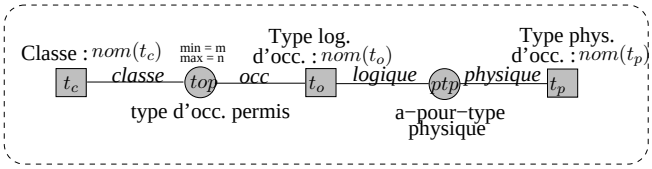


FIG. 2.23 – Le schéma type d'une définition utilisant la primitive `type d'occurrence permis`.

Une association *csc* de type `classe-sousclasse` au sein de laquelle un topic t_{c_1} joue le rôle `superclasse` et un topic t_{c_2} joue le rôle `sousclasse`. Les associations de type `classe-sousclasse` définissent un ordre partiel sur les topics et permettent l'héritage des `types d'association permis`, des `types d'occurrence permis` et de leurs cardinalités respectives. La figure 2.24 présente le schéma type d'une définition avec `classe-sousclasse`.

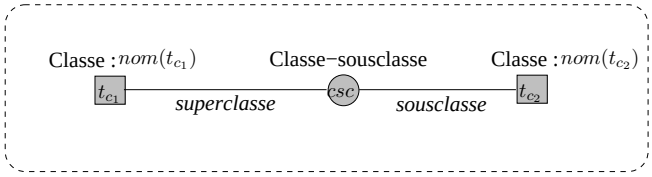


FIG. 2.24 – Le schéma type d'une définition utilisant la primitive `classe-sousclasse`.

Associations orientées

On peut définir dans une ontologie ITM un type particulier d'associations : les types d'`association orientée`.

Un type d'`association orientée` t_a est un type d'association conventionnel relié par `type d'association orientée permis` à un topic t_r jouant un rôle de type soit `prédécesseur`, soit `successeur`, ainsi qu'à un topic c_t représentant la classe du topic t qui peut jouer le rôle de type t_r dans une association de type t_a au niveau instance. La figure 2.25, présente le schéma type de définition d'une association orientée au niveau modèle.

Le topic t_a de type `type d'association` peut porter une occurrence h de type logique `type de hiérarchie` qui permet de préciser la nature du réseau que les associations de

t_a doivent former au niveau instance. Il existe sept `types de hiérarchie` dans ITM : le `graphe sans circuit`, l'`arbre` et l'`arborescence` ; ainsi que leurs versions `transitives` auxquelles on ajoute l'`association transitive`.

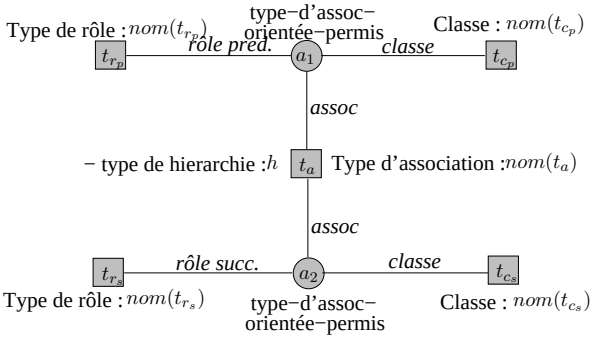


FIG. 2.25 – Le schéma type d'une définition utilisant la primitive `type d'association orientée permis`.

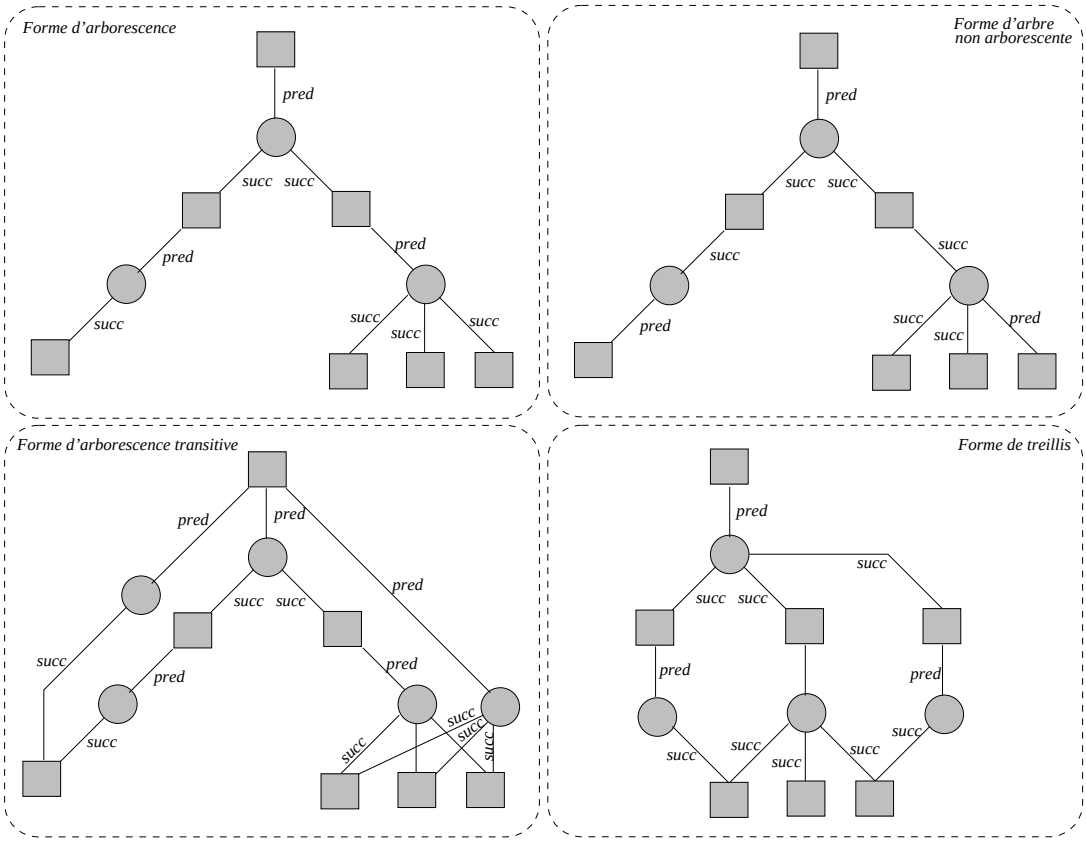


FIG. 2.26 – Quatre TM respectivement sous forme d'`arborescence`, d'`arbre`, d'`arborescence transitive` et de `graphe sans circuit` (les types et les noms sont absents des figures).

Afin de définir l'impact de ces sept `types de hiérarchie` sur le réseau induit par un `type d'association orienté` t_a donné, on définit dans la suite des notions de *cycle* et de *circuit* de topics en s'appuyant sur deux fonctions : $\mathcal{V}_{t_a}$ et $\mathcal{P}_{t_a}$.

$\mathcal{V}_{t_a}$ attribue à un topic t l'ensemble des topics reliés à t dans la TM par un chemin exclusivement constitué d'associations de type t_a. Il en est de même pour $\mathcal{P}_{t_a}$; à ceci près que l'ensemble associé à t ne contient que des *prédécesseurs* de t. Les définitions de $\mathcal{P}_{t_a}$ et $\mathcal{V}_{t_a}$ s'appuient sur deux fonctions auxiliaires p_{t_a} et v_{t_a} qui associent à un topic de l'ensemble T d'une TM un sous-ensemble de T.

L'ensemble $p_{t_a}(t)$ des prédécesseurs directs d'un topic t par rapport à t_a est l'ensemble des topics jouant un rôle caractérisé de `prédécesseur` dans toutes les associations de type t_a au sein desquelles t joue un rôle `successeur`. L'ensemble $v_{t_a}(t)$ des voisins directs de

t contient les topics participant aux associations de type t_a au sein desquelles t joue un des deux rôles.

L'ensemble $\mathcal{P}_{t_a}(t)$ des prédécesseurs d'un topic t par rapport à t_a est déterminé en recherchant récursivement les prédécesseurs directs de chaque topic de l'ensemble retourné par p_{t_a} jusqu'à stabilisation du résultat. L'ensemble $\mathcal{V}_{t_a}(t)$ des topics en relation par t_a avec t est déterminé de la même façon par récurrence sur v_{t_a}.

Dans une TM, les associations de `type d'association orienté` t_a forment un *cycle* ssi il existe un topic $t \in T$ qui appartienne à $\mathcal{V}_{t_a}(t)$. En plus de former un cycle, elles forment un *circuit* ssi t appartient à $\mathcal{P}_{t_a}(t)$. De plus, on dit que le sous-graphe induit par ces associations est *transitivement fermé* ssi pour tout topic t de T et tout topic t' dans $\mathcal{P}_{t_a}(t)$, t' est dans $p_{t_a}(t)$.

Lorsque l'occurrence h de type logique `type de hiérarchie` que porte le topic t_a a pour valeur `graphe sans circuit`, `arbre` et `arborescence`, leurs versions transitives `graphe sans circuit transitif`, `arbre transitif`, `arborescence transitive` ; et enfin `association transitive`, le sous-graphe tm_{t_a} d'une TM donnée induit par les associations de type t_a doit respecter certaines conditions :

- lorsque la valeur de h est `graphe sans circuit`, tm_{t_a} doit être sous forme de graphe sans circuit : c'est-à-dire que les associations de tm_{t_a} ne doivent former aucun circuit ;
- lorsque la valeur de h est `arbre`, il doit être sous forme d'arbre : c'est-à-dire que les associations de tm_{t_a} ne doivent former aucun cycle ;
- enfin, lorsque la valeur de h est `arborescence`, il doit être sous forme d'arborescence : c'est-à-dire qu'il doit être sous forme d'arbre et que pour tout topic t l'ensemble $p_{t_a}(t)$ ne doit contenir qu'un seul topic.

Lorsque la valeur de h est `graphe sans circuit transitif`, `arbre transitif` ou `arborescence transitive`, un tel sous-graphe tm_{t_a} doit être sous forme de `graphe sans circuit transitif`, d'`arbre transitif` ou d'`arborescence transitive` ssi il forme respectivement un `graphe sans circuit`, un `arbre` ou une `arborescence` et qu'en plus il est *transitivement fermé*.

Quand h a pour valeur `association transitive`, il doit être transitivement fermé mais n'est pas forcément sous forme de `graphe sans circuit`, d'`arborescence`, ou d'`arbre`.

La figure 2.26 présente un exemple pour quatre des sept types de hiérarchie possibles.

2.4.3 Espaces de travail

Le niveau *instance* est découpé en "espaces de travail" dédiés à un type de connaissance : annotation sémantique, thésaurus, structure organisationnelle des documents... Chaque espace de travail E_x est défini à partir d'une TM tm_x et permet l'usage de *métadonnées* pour exprimer des informations sur les éléments de tm_x. Ces métadonnées

sont stockées dans le SGBD d'ITM et n'appartiennent pas au formalisme TM. Elles permettent par exemple d'indiquer l'auteur ou la date de création ou de modification d'un élément TM de base. L'auteur d'une création peut être un agent humain (un utilisateur identifié par un nom de compte et un mot de passe) ou logiciel, et la métadonnée *auteur* permet d'établir cette différence. Le vocabulaire utilisé dans la TM tm_x de chaque espace E_x au niveau instance est défini dans la TM tm_{modele} du niveau modèle.

Les espaces respectent les contraintes suivantes :

- Les espaces E_{meta} et E_{modele} sont respectivement définis à partir de la TM complète tm_{meta} du niveau *méta* et de la tm_{modele} du niveau *modèle*, tandis que la TM $tm_{instances}$ du niveau *instances* rassemble les TM des espaces de travail ;
- tous les topics des TM des niveaux méta et modèle sont identifiés par un nom (i.e. la fonction *nom* est totale pour ces TM) ;
- les références aux types utilisées dans la TM associée à un espace respectent la hiérarchie des niveaux, *i.e.* $RefType(tm_{modele}) \subseteq Nom(tm_{meta})$ et pour chaque espace E_x, $RefType(tm_x) \subseteq Nom(tm_{modele})$.
- La TM tm_x associée à un espace E_x est disjointe de celles associées aux autres espaces ; c'est à dire qu'aucune TM incluse dans tm_x n'est incluse dans une TM associée à une autre espace. Il faut noter que les pointeurs, qui n'appartiennent pas au formalisme standard des TM, ne sont pas concernés par cette condition. Les pointeurs permettent de créer des liens entre topics d'espaces différents. Comme le montre la figure 2.18, on peut par exemple associer un terme particulier du thesaurus à une ressource de l'espace d'annotation. La TM *tm* d'un espace peut présenter des pointeurs établissant une référence vers le nom d'un topic qui n'appartient pas à *tm*.

L'espace dédié aux annotations est désigné par $E_{\mathcal{A}}$ et la TM $tm_{\mathcal{A}}$ qui lui est associée contient l'ensemble des annotations d'ITM.

2.4.4 Comparaison avec les autres formalismes

L'ensemble des formalismes présentés précédemment se divise essentiellement en deux catégories :

- ceux qui modélisent une TM en théorie des graphes : c'est à dire le format *TMG* (voir section 2.3.4) et le format à base d'hypergraphes (présenté en section 2.3.5) ;
- et ceux qui conservent au mieux la structure du document XTM : c'est à dire le format *TMDM* (voir section 2.3.1) et le formalisme des *Subject Maps* (voir section 2.3.2) ;

le modèle Q (voir section 2.3.3) étant à part (et constituant une catégorie à lui seul).

En effet, l'interprétation d'un document *XTM* (plus précisément *XTM 2.0*) en *TMDM* ou *Subject Maps* préserve presque toujours l'emboîtement des éléments *XML* dans la structure de l'objet résultant, alors que dans un document interprété graphiquement,

les éléments *XML* sont éclatés dans le réseau et les objets obtenus (des sommets) n'ont aucune profondeur.

Le formalisme $XTMG_{MDK}$ se range dans la première catégorie dite *graphique*. Il a été conçu pour épouser les spécificités du format XTM_{MDK}. Tout graphe expressible dans le format XTM_{MDK} est donc définissable en $XTMG_{MDK}$. La réciproque aussi est vérifiée : tout graphe définissable dans le format $XTMG_{MDK}$ est expressible en XTM_{MDK}. Le format XTM_{MDK} étant une version du format *XTM* légèrement plus contrainte, certaines TM expressibles en *XTM 1.0* le sont plus difficilement en XTM_{MDK} et, par voie de conséquence, en $XTMG_{MDK}$.

La différence essentielle que $XTMG_{MDK}$ entretient avec les autres formalismes - à part celui à base d'hypergraphes - réside en ce qu'il attribue un statut particulier à certains éléments XML du format *XTM 1.0* en les modélisant sous la forme de fonctions et en les distinguant des composants du réseau. En *Subject Maps*, en *TMDM* et en *Q* le moyen d'attribuer un nom, un type, un domaine de validité ou un sujet est de même nature que l'établissement des relations entre entités.

La démarche utilisée pour $XTMG_{MDK}$ peut être rapprochée de celle entreprise dans [AdMRV02] (voir section 2.3.5) pour modéliser les TM par des hypergraphes. Dans cette approche, certains éléments XML du format *XTM 1.0* sont représentés par la fonction θ ; cette fonction étant utilisée pour *stratifier* l'ensemble des TM. Il faut remarquer que cette fonction θ partage plusieurs significations selon son utilisation. Elle peut attribuer un domaine de validité à un élément de la TM, ou le typer. Dans le formalisme $XTMG_{MDK}$, les éléments XML du format XTM_{MDK} permettant d'attribuer un nom, un type, un domaine de validité ou un sujet sont représentés par quatre fonctions différentes, ce qui permet de conserver la distinction sémantique. Il faut souligner que puisque cette information est perdue lors de la modélisation d'une TM dans le format à base d'hypergraphes, la réversibilité n'est plus possible car il existe plusieurs documents *XTM 1.0* pour un hypergraphe TM. Par conséquent il peut exister plusieurs graphes $XTMG_{MDK}$ correspondant à un hypergraphe TM.

Cette non réversibilité de la représentation existe aussi chez le format *TMG*. Ce dernier permet de modéliser toutes les TM représentables dans le format standard *1.0*, cependant il est possible de créer un graphe *TMG* qui n'a pas d'équivalent en *XTM 1.0* ou en XTM_{MDK}. De tels graphes ne sont pas expressibles en $XTMG_{MDK}$.

Chapitre 3

La famille $\mathcal{SG}$

Le modèle des graphes conceptuels (GC) a été introduit dans les années 80 par John F. Sowa [Sow84]. Il appartient à la famille des réseaux sémantiques et permet de représenter des connaissances de manière graphique. Différents usages sont associés aux GC selon les approches. Les GC proposés par John F. Sowa sont utiles pour donner une forme graphique à des expressions de la logique des prédicats sans pour autant que leur pouvoir d'expression ne se limite à un fragment particulier de cette logique. La famille $\mathcal{SG}$ des graphes conceptuels se conforme davantage aux exigences d'une utilisation pratique des GC. L'approche consiste à étudier en quelle mesure les GC peuvent offrir des services concrets tout en gardant un pouvoir d'expression maximum, dusse ce dernier être amoindri afin de garantir un usage convenable. Cette approche conduit à étudier et cibler des fragments décidables de la logique des prédicats pour donner une base formelle à la sémantique des connaissances expressibles et des raisonnements possibles dans la famille $\mathcal{SG}$. Les GC et la famille $\mathcal{SG}$ présentent les caractéristiques suivantes :

- Comme énoncé précédemment, ils possèdent une sémantique formelle en logique des prédicats.
- La famille $\mathcal{SG}$ fournit différents types de connaissances : des faits représentés par des graphes conceptuels simples, des règles d'inférence, ainsi que des contraintes.
- Les raisonnements sont basés sur des opérations de graphe ; ce qui fournit potentiellement des possibilités d'explication des raisonnements à l'utilisateur puisque les raisonnements peuvent être visualisés dans le formalisme que celui-ci connaît et directement sur les connaissances qu'il a définies. Ces raisonnements sont corrects et complets par rapport à la sémantique formelle.
- Les algorithmes développés pour les graphes conceptuels, et en particulier pour la famille $\mathcal{SG}$, utilisent des techniques de combinatoire (notamment de théorie des graphes et de satisfaction de contraintes) qui les rendent particulièrement efficaces.
- La famille $\mathcal{SG}$ est implémentée dans CoGITaNT [Gen97], une API de développe-

ment d'applications à base de graphes conceptuels, utilisable librement [Gen97], et bénéficie d'outils graphiques associés TooCOM [Fur04] et CoGUI [Gut06].

Cette section présente la partie de la famille $\mathcal{SG}$ utilisée dans ce travail de thèse. Pour une présentation complète et une étude approfondie de sa complexité, voir [BM02] et [CM92].

3.1 Les graphes simples.

Un *graphe conceptuel simple (SG)* est un graphe biparti étiqueté dont l'une des classes de sommets, dite de sommets *concepts*, représente des entités, et l'autre, dite de sommets *relations*, représente des relations entre ces entités ou des propriétés de ces entités.

Le graphe G de la figure 3.1 peut être considéré comme représentant la connaissance suivante : "deux chercheurs, dont un travaille avec K dans le bureau 24, sont membres d'un projet auquel participe le chercheur J dont le bureau est situé près du bureau 24".

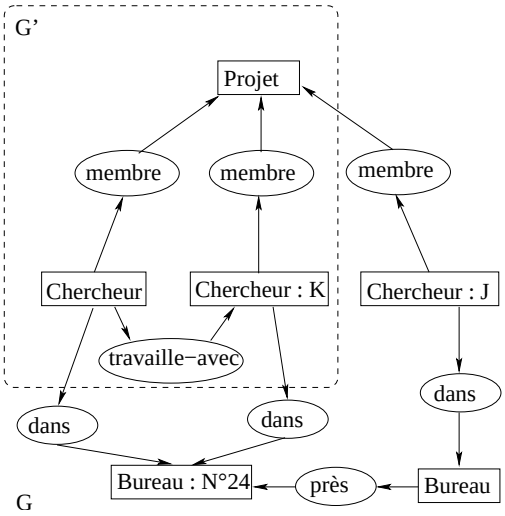


FIG. 3.1 – Un exemple de SG.

Les étiquettes des sommets sont définies dans un vocabulaire appelé *support* qui peut être plus ou moins riche. Le support peut être considéré comme une ontologie rudimentaire et les SG encodent des connaissances assertionnelles (que nous appellerons des *faits*) : ils assertent l'existence d'entités et de relations entre ces entités.

3.1.1 Support

Nous considérerons ici un support comme une structure $S = (T_C,\ T_R,\ I,\ \sigma)$.

T_C est un ensemble de types de concepts et T_R est un ensemble de types de relations d'arité (nombre d'arguments) quelconque. T_R est partitionné en k sous-ensembles disjoints T_R^1, T_R^2, ... T_R^k, chacun contenant respectivement les types de relations d'arité 1 à k. T_C et chaque T_R^i sont partiellement ordonnés, l'ordre partiel traduisant une relation de spécialisation ($t' \leq t$ s'interprète par "t' est une spécialisation de t").

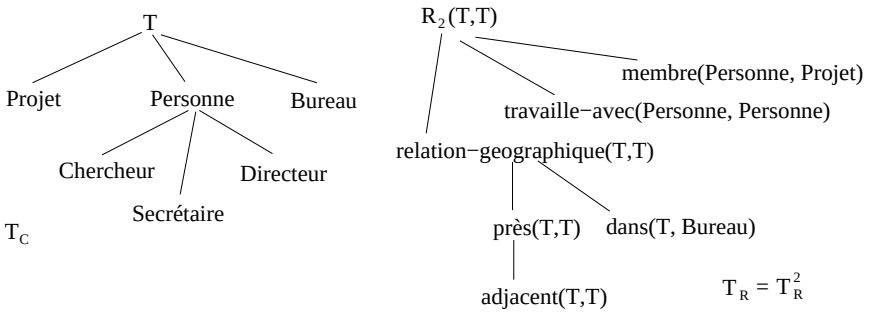


FIG. 3.2 – Un exemple de support.

I est un ensemble de marqueurs dits *individuels*. On dispose en plus d'un marqueur *générique* noté $*$ différent des éléments de I. L'ordre entre les éléments de $I \cup \{*\}$ est tel que deux éléments de I sont incomparables et le marqueur $*$ est le plus grand des éléments.

σ associe à toute relation une signature qui définit le type maximal de chacun de ses arguments.

Le support peut en plus disposer d'une application τ dite de *conformité* de I dans T_C qui permet d'associer à un marqueur individuel un type de concept unique.

Dans la figure 3.2, le type de concept `Chercheur` est plus spécifique que `Personne`, et le type de relation `relation-géographique` est plus général que le type `dans` dont la signature impose au second argument d'être un concept de type `Bureau` et permet au premier d'être de type quelconque.

3.1.2 Graphe simple

Un graphe simple (SG) G défini à partir d'un support S est un multigraphe[1] biparti fini $G = (C_G, R_G, E_G, l_G)$ où C_G est l'ensemble des sommets concepts, R_G l'ensemble des sommets relations. E_G est l'ensemble des arêtes multiples entre C_G et R_G. Les arêtes incidentes à un sommet relation sont totalement ordonnées de 1 jusqu'au degré du sommet. l_G est la fonction d'étiquetage des sommets et des arêtes.

[1] graphe pouvant présenter de multiples arêtes entre sommets

Un sommet concept est étiqueté avec l_G par un couple $t : m$ où t est un type de concept de T_C, m est un marqueur appartenant à $I \cup \{*\}$. Si $m \in I$ le concept est dit *individuel* et $t \leq \tau(m)$. Si le sommet désigne une entité non identifiée, il est dit *générique* et est représenté par le marqueur *générique* noté $*$.

Un sommet relation r est étiqueté avec l_G par un type de relation t_r de l'ensemble T_R. Le degré de r doit être égal à l'arité de son type. Si n est l'arité de t_r, n arêtes (t_r, i, c) (avec i de 1 à n) sont incidentes à au sommet r et ordonnées selon i. En d'autres termes, si $t_r \in T_R^k$ alors $|\{(r, i, c)|(r, i, c) \in E_G\}| = k$ et $\{i|(r, i, c) \in E_G\} = \{1, .., k\}$.

Dans les figures, les sommets concepts sont représentés par des rectangles et les sommets relations par des ovales, et chaque arête (r, i, c) entre un sommet concept c et un sommet relation r n-aire est représentée par un segment entre c et r étiqueté par le nombre i. Lorsque les relations sont binaires, on a coutume de remplacer les arêtes respectivement incidentes au premier et second voisin par un arc entrant et un arc sortant incidents à la relation. En général, on ne fait pas apparaître le marqueur générique $*$. Dans le graphe G de la figure 3.1 par exemple, le sommet `[Chercheur :K]` désigne "le" chercheur K, tandis que le sommet `[Chercheur]` désigne "un" chercheur.

3.1.3 Projection

On définit une relation de spécialisation/généralisation sur les SG qui peut être caractérisée par une sorte d'appariement de graphes (un homomorphisme de graphes), appelé *projection*. Intuitivement, l'existence d'une projection de G dans H montre que la connaissance représentée dans G est contenue dans H, autrement dit que G se déduit de H, ou encore que H est une spécialisation de G (noté $H \leq G$), ou bien que G est plus général que H (noté $G \geq H$).

Une projection π de G dans H est plus précisément une application de C_G dans C_H et de R_G dans R_H qui conserve les arêtes et peut spécialiser les étiquettes des sommets :

- pour toute arête (r, i, c) dans E_G, $(\pi(r), i, \pi(c))$ est dans E_H
- pour tout sommet x de $C_G \cup R_G$, $l_H(\pi(x)) \leq l_G(x)$

On dit de G et H qu'ils sont respectivement les graphes *source* et *cible* de la projection.

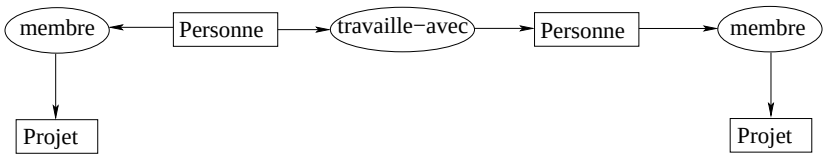


FIG. 3.3 – Un exemple de SG pouvant être considéré comme une requête.

Si on considère les figures 3.3 et 3.1, il existe une projection de Q dans G (sachant que `Chercheur < Personne`). Le sous-graphe G' entouré de pointillés est l'image de cette

projection.

Étant donnés deux SG G et H définis selon un support S, le problème qui consiste à savoir s'il existe une projection de G dans H est classé parmi les problèmes NP-complets.

Notion de requête

Considérons maintenant une base de connaissances composée d'un support et d'une base F d'assertions (ou faits). Cette base peut être interrogée par le mécanisme de projection. Sous sa forme la plus simple, une requête, est elle-même un SG. Par exemple, traiter le SG Q de la figure 3.3 comme une requête, consiste à rechercher les occurrences du motif suivant : "deux chercheurs ayant un bureau et travaillant ensembles". La base répond à la requête s'il existe une projection de Q dans F, ce qui se traduit par le fait que Q se déduit de la base de connaissances.

3.1.4 Relations avec la logique

Ce formalisme de base est muni d'une sémantique en logique du premier ordre par le biais d'une application notée Φ [Sow84]. Un support et un SG se traduisent par Φ en formules de la logique des prédicats. La formule associée à un SG est existentielle, positive et conjonctive tandis que celles associées à un support sont des implications universellement quantifiées. Le résultat de correction [Sow84] et de complétude [CM92] de la projection établit l'équivalence entre l'existence d'une projection et la déduction logique sur les formules associées aux SG.

Soit un support $S = (T_C, T_R, I, \sigma)$, à chaque marqueur individuel de I est associée une constante, et à chaque type de T_R (resp. T_C) est associé un prédicat de même arité k (resp. d'arité 1) et de même nom que le type considéré. $\phi(S)$ fournit un ensemble de formules telles que pour tout type t' plus spécifique qu'un autre type t, on ait la formule $\forall x_1 ... x_k (t'(x_1, ..., x_k) \rightarrow t(x_1, ..., x_k))$, ou k est l'arité de t et t'.

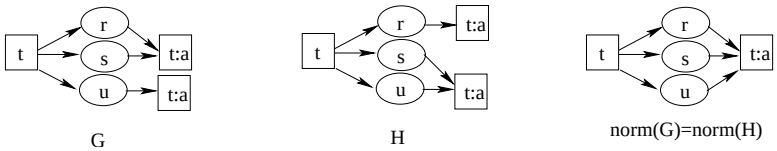
Soit un SG $G = (C_G, R_G, E_G, l_G)$, la formule $\phi(G)$ est obtenue de la manière suivante. Pour chaque sommet concept c, si c est générique on lui associe une variable différente de toutes celles associées aux autres sommets génériques, sinon on lui associe la constante correspondant au marqueur individuel. Un atome $t(c)$ (resp. $t(c_1, ..., c_k)$) est ensuite associé à chaque sommet concept (resp. sommet relation r d'arité k) où t est le type du sommet concept et c (resp. c_i) la variable ou constante attribuée au marqueur de ce sommet (resp. attribuée au $i$ème voisin de r). $\phi(G)$ est la fermeture existentielle de la conjonction des atomes précédemment obtenus.

La formule associée au graphe G_1 de la figure 3.1 est par exemple : $\Phi(G) = \exists x \exists y \exists z (Chercheur(x) \wedge Projet(y) \wedge Bureau(z) \wedge Chercheur(K) \wedge Chercheur(J) \wedge Bureau(N24) \wedge membre(x, y) \wedge membre(K, y) \wedge membre(J, y) \wedge travaille - avec(x, K) \wedge pres(z, N24) \wedge dans(J, z) \wedge dans(K, N24) \wedge dans(x, N24))$.

La projection est correcte[2] par rapport à la déduction logique, c'est-à-dire que si un graphe H contient (au sens de la projection) la connaissance exprimée dans un graphe G, alors son correspondant logique $\phi(G)$ se déduit de $\phi(H)$ et $\phi(S)$. En d'autres termes, si G et H sont deux SG définis sur un support S, tels que $H \leq G$ alors $\phi(S), \phi(H) \models \phi(G)$.

La complétude de la projection avec la déduction logique s'obtient à condition que le graphe cible soit sous forme normale. La forme normale d'un SG G est le SG $norm(G)$ obtenu en fusionnant les sommets de G qui ont un même marqueur individuel. Ce SG peut être obtenu en temps linéaire et on est assuré de son existence lorsque les sommets de G respectent la condition de conformité (ce qui est supposé dans ce travail). La condition de complétude de la projection par rapport à la déduction logique peut s'énoncer de la manière suivante : si G et H sont deux SG définis sur un support S, tels que $\phi(S), \phi(H) \models \phi(G)$ alors $H \leq G$.

La figure 3.4 présente un exemple d'incomplétude d'une projection dans un graphe non normalisé. En effet, le graphe G se projette dans $norm(G)$ mais pas dans H, alors que $\phi(G)$ se déduit logiquement de $\phi(H)$ puisque ces trois graphes ont le même correspondant logique par ϕ (ils sont équivalents).



$$\phi(G) \equiv \phi(H) \equiv \phi(norm(G)) \equiv \exists(xt(x) \wedge t(a) \wedge r(x,a) \wedge s(x,a) \wedge u(x,a))$$

FIG. 3.4 – Exemple de l'incomplétude d'une projection dans un graphe non normalisé.

3.1.5 Irredondance

Deux SG sont dits *équivalents* s'ils se projettent l'un dans l'autre. On dit qu'un SG est *redondant* s'il est équivalent à l'un de ses sous-graphes stricts (c'est à dire à un de ses sous-graphes différents de lui même), sinon il est *irredondant*. Intuitivement, un SG est estimé redondant lorsqu'on peut réduire le nombre de ses sommets sans en changer sa signification. Les SG construits sur un support donné peuvent être regroupés en classes d'équivalence et chacune d'entre elles admet un unique graphe irredondant. Notons que le problème qui consiste à savoir si un SG est redondant est NP-complet. La *forme irredondante* d'un SG G est un sous-graphe irredondant de G qui lui est équivalent[3].

[2] selon la correspondance établie par la fonction ϕ entre la projection et la déduction logique

[3] Les sous-graphes de G irredondants et équivalents à G sont isomorphes.

3.1.6 La notion d'*égalité* et de *différence*

Deux sommets concepts dans un SG ne désignent pas forcément des entités identiques ou différentes. Pour exprimer ce genre de propriétés, la famille $\mathcal{SG}$ dispose de deux liens dits de *coréférence* et de *différence*.

Le lien de *coréférence*

S'il est nécessaire d'exprimer dans un SG que deux sommets concepts différents désignent une même entité, on dispose d'une relation binaire particulière définie sur l'ensemble des sommets concepts. Cette relation est réflexive, symétrique et transitive ; et sa sémantique logique est celle de $=$. Elle est notée $coref$, prend le nom de lien de *coréférence* et est représentée dans les figures par un segment en pointillés reliant les deux sommets concepts sur lesquels elle s'applique. Une projection π de G dans H prend en compte le lien de coréférence de telle sorte que si $\{c_1, c_2\} \in coref_G$ alors $\{\pi(c_1), \pi(c_2)\} \in coref_H$. Deux sommets génériques c_1 et c_2 reliés par un lien de coréférence, sont traduits en logique par une même variable. Si l'un d'eux est individuel, il sont traduits en la constante associée au marqueur du concept individuel.

Le lien de *différence*

Deux sommets concepts différents d'un SG ne désignent pas forcément des entités différentes. Pour exprimer la différence, on peut ajouter aux SG une relation binaire particulière sur l'ensemble des sommets concepts : elle est antiréflexive et symétrique, et sa sémantique logique est celle de $\neq$. Cette relation, notée dif, est en général représentée par un segment étiqueté $\neq$ reliant deux sommets concepts et appelé "lien de différence". Nous faisons l'hypothèse classique du nom unique (*unique name assumption*), toute paire de sommets concepts ayant des marqueurs individuels différents appartient donc à la relation dif. Une projection π de G dans H doit alors prendre en compte la différence : si $\{c_1, c_2\} \in dif_G$ alors $\{\pi(c_1), \pi(c_2)\} \in dif_H$. Dans le cas général, la projection, même ainsi étendue, n'est plus complète par rapport à la déduction en logique classique car le principe du tiers-exclu intervient, et on considère alors qu'étant données deux entités, soit elles sont égales, soient elles sont différentes. Pour déterminer si G se déduit de H, il ne suffit plus de tester s'il existe une projection de G dans H, mais s'il en existe une dans H^c, pour tout graphe "complet" H^c que l'on peut générer à partir de H en précisant pour chaque paire de sommets concepts s'ils sont égaux ou différents.

L'incomplétude de la projection ne se pose plus si l'on considère une logique qui n'adopte pas le principe du tiers-exclu, comme la logique intuitionniste [LM06]. En effet, une telle logique ne permet pas de déduire une différence à partir de l'absence de différence puisque le tiers-exclu n'est plus adopté. Il faut donc que la relation de différence apparaisse explicitement au cours d'une déduction ou d'une assertion. Dans ce cas, on a une

projection π de G dans H telle que pour tout $\{c_1, c_2\} \in dif_G$ on ait $\{\pi(c_1), \pi(c_2)\} \in dif_H$ ssi $\phi(S), \phi(H) \models \phi(G)$ puisqu'on peut déduire l'expression $\phi(c_1) \neq \phi(c_2)$ de $\phi(G)$ à partir de l'assertion $\phi(\pi(c_1)) \neq \phi(\pi(c_2))$ de $\phi(H)$. Dans la figure **??**, il n'existe "naturellement" aucune projection du graphe G dans le graphe H puisque les sommets différents de G ne se projettent pas sur les sommets de H dont on ne peut savoir s'ils sont différents.

3.1.7 Forme bicolorée

Les SG sont les constituants de base à partir desquels sont élaborés les autres éléments de la famille $\mathcal{SG}$. Les règles et les contraintes SG présentées dans les sections suivantes sont définies sous la forme de SG bicolorés.

Un SG bicoloré est un couple $K = (G, \rho)$ où G est un SG et ρ est une fonction totale de $C_G \cup R_G$ dans $\{0, 1\}$ appelée *fonction de coloration* des sommets concepts et relation. Le nombre 1 ou 0 associé au sommet est la *couleur* du sommet. On note $K_{(i)}$ le sous-graphe de G (1) induit par les sommets de couleur i et (2) formant un SG (c'est à dire que les voisins d'un sommet relation de $K_{(i)}$ appartiennent aussi à $K_{(i)}$ même s'ils ne sont pas de couleur i). Les sommets concepts qui appartiennent à la fois à $K_{(0)}$ et $K_{(1)}$ sont appelés *sommets frontières* . Ils sont toujours de couleur 0 et sont reliés à au moins un sommet relation de couleur 1.

3.2 Les règles $\mathcal{SG}$.

Au formalisme "noyau" des graphes conceptuels simples, la famille $\mathcal{SG}$ ajoute deux types de connaissances représentées sous forme bicolorée : les *règles* et les *contraintes*. Cette section présente les *règles* SG et les mécanismes d'inférence à partir desquelles les connaissances sont déduites. Une règle SG R qui exprime une connaissance de la forme "si hypothèse alors conclusion" est un SG bicoloré dont les sous-graphes $R_{(0)}$ et $R_{(1)}$ représentent respectivement l'hypothèse et la conclusion.

La figure 3.5 présente deux exemples de règles. L'hypothèse de la règle se compose des sommets blancs (couleur 0) alors que sa conclusion est coloriée en gris (couleur 1). Les règles R_1, R_2 et R_3 expriment les propriétés des relations `proche` et `membre`. R_1 donne son caractère symétrique à la relation `proche` tandis que R_2 permet de déduire la proximité de deux bureaux s'ils sont adjacents à un même troisième. R_3 exprime que chaque chercheur est membre d'un projet.

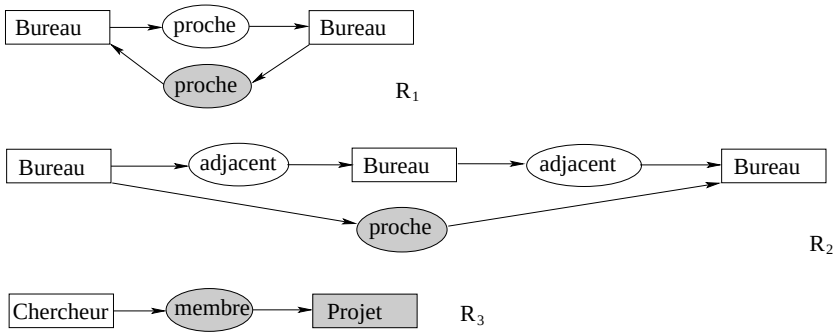


FIG. 3.5 – Règles SG

Afin de permettre la représentation du lien de coréférence dans une règle définie par un SG bicoloré, on contraint ce dernier à prendre une coloration qui indique son appartenance ou non à la partie conclusion. Pour colorier ce lien il faut respecter la règle suivante : tout lien de coréférence relié à un sommet non frontière de la conclusion prend la couleur de ce sommet ; et par conséquence appartient à la conclusion. Il faut remarquer que les règles SG peuvent aussi être définies sous la forme de deux SG disjoints (l'un représentant l'hypothèse et l'autre la conclusion) dont les sommets frontières sont reliés par des liens de *coréférence* (cette définition n'est pas présentée ici pour plus de détails voir [BM02]).

3.2.1 Application d'une règle et dérivation

Une règle R s'applique à un SG F s'il existe une projection de l'hypothèse de R dans F. L'application de R à F selon une telle projection π consiste à "attacher" à F la conclusion de R, en fusionnant chaque sommet frontière de la conclusion avec l'image par π du sommet frontière lui correspondant dans l'hypothèse. Notons que ce mécanisme est complètement basé sur des opérations de graphe.

Plus formellement : soit F un SG et R une règle, R est applicable à F s'il existe une projection π de $R_{(0)}$ dans F. Dans ce cas, le résultat de l'application de R dans F relativement à π est le SG F' obtenu en effectuant l'union disjointe de F avec $R_{(1)}$, puis en fusionnant chaque sommet frontière s de $R_{(1)}$ avec $\pi(s)$. On dit de F' qu'il est la R-dérivation directe de F.

La figure 3.6 présente deux applications des règles R_2 et R_1 de la figure 3.5. F_1 et F_2 sont respectivement obtenus en appliquant R_2 sur F_0 et F_1 et sont respectivement une R_2-dérivation et une R_1-dérivation directe de F_0 et de F_1. Les sommets ajoutés lors de chaque application sont dans la portion grise.

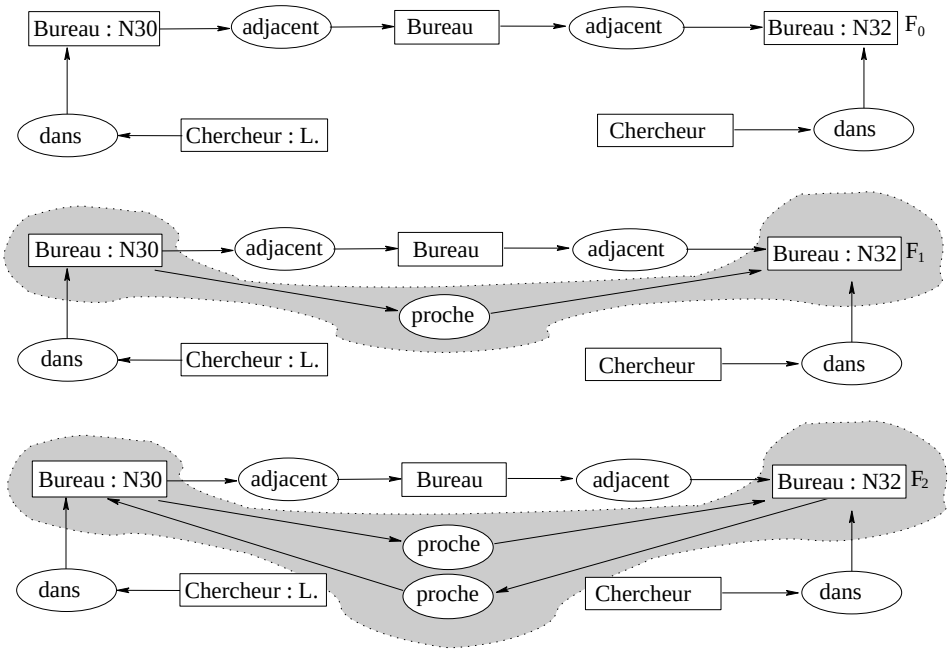


FIG. 3.6 – Applications de règle

Étant donné un fait F et un ensemble de règles $\mathcal{R}$, on dit qu'un SG F' se dérive de F par $\mathcal{R}$ s'il existe une séquence d'applications de règles de $\mathcal{R}$ menant de F à F'. Plus précisément, la $\mathcal{R}$-dérivation de F à F' est une séquence de SG $F = F_0, ..., F_k = F'$ telle que pour tout $1 \leq i \leq k$, F_i soit une R-dérivation directe de F_{i-1} et R une règle de $\mathcal{R}$.

On considère maintenant une base de connaissances $K = (S, F, \mathcal{R})$ composée d'un support S , d'un ensemble de règles $\mathcal{R}$ et d'un fait F définis par rapport au support. Le mécanisme d'interrogation de K doit prendre en compte la connaissance implicite encodée dans les règles. La base de connaissances répond à une requête Q s'il existe un SG F' dérivé de F (par les règles de $\mathcal{R}$) tel que Q se projette dans F'.

Considérons par exemple la figure 3.7 qui présente un graphe F' obtenu par $\mathcal{R}$-dérivation à partir du graphe F_0 de la figure 3.6. Les sommets ajoutés par application de règles ont leurs contours en pointillés. On peut établir une projection entre le SG Q de cette figure et le sous-graphe F'' délimité par la portion grise. Si Q est vue comme une requête, l'image F'' de Q peut être considérée comme une des réponses déductibles à partir de F_0 et des règles $\mathcal{R}$.

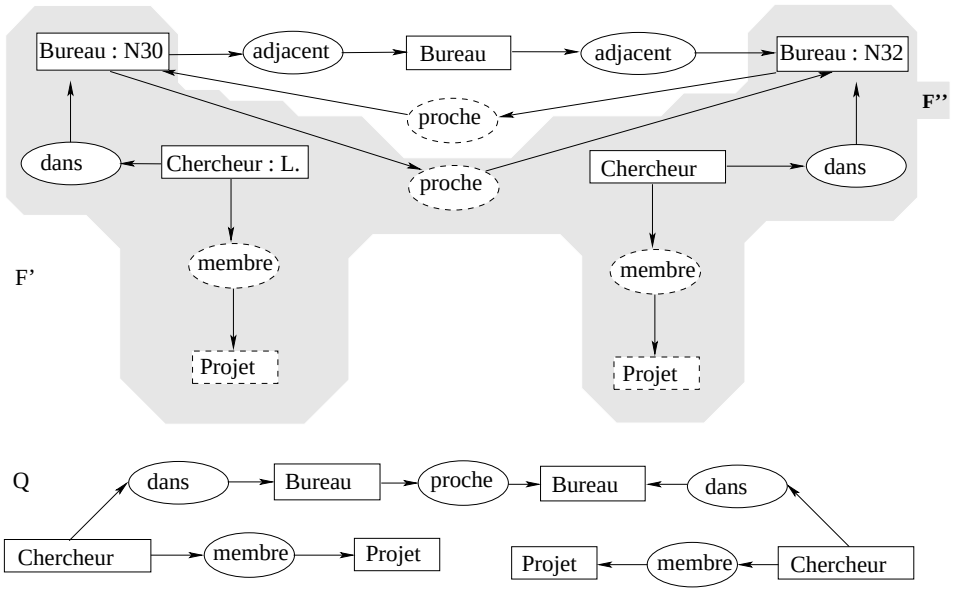


FIG. 3.7 – Réponse à une requête après $\mathcal{R}$-dérivation

Pour déterminer si un tel SG Q se déduit de la base de connaissances K, on dispose de mécanismes de chaînage avant et de chaînage arrière adéquats et complets par rapport à la déduction logique [SM96]. Précisons que le problème d'interrogation n'est plus décidable (autrement dit il n'existe pas d'algorithme qui résolve ce problème en un temps fini quelles que soient les données) si l'on ne restreint pas la forme des règles.

Règles à dérivation finie

Néanmoins, lorsque les règles sont à dérivation finie, la décidabilité du problème est assurée quelle que soit la base de faits. Ces règles se définissent à partir de la notion de graphe *complet* : un graphe irredondant H est dit complet par rapport à un ensemble de règles $\mathcal{R}$ lorsque tout graphe résultant de l'application sur H d'une de ces règles est équivalent à H.

Un ensemble de règles est dit à dérivation finie si pour tout SG G, il existe une $\mathcal{R}$-dérivation G ... G' telle que $irr(G')$ soit *complet* par rapport à $\mathcal{R}$. Le graphe complet est noté $G^{\mathcal{R}}$.

Pour un tel ensemble $\mathcal{R}$ de règles, le problème de déduction pour une base de connaissances $K = (S, G, \mathcal{R})$ devient décidable. Pour déterminer si un SG Q est déductible de K, il suffit d'obtenir $G^{\mathcal{R}}$ et de vérifier l'existence d'une projection de Q dans $G^{\mathcal{R}}$ [4].

[4] Lorsque les conclusions de règles ont des sommets individuels, il est néanmoins nécessaire de mettre chaque graphe résultant d'une application de règle sous forme normale

Toutefois, cette condition n'est pas nécessaire et pour certains ensembles de règles la déduction peut être décidable sans qu'ils soient à dérivation finie.

Il existe des conditions suffisantes qui permettent de déterminer si un ensemble de règles est à dérivation finie. L'une d'entre elles, énoncée dans [BS06], stipule qu'un ensemble de règles est tel si aucun cycle de dépendance n'apparaît au sein de l'ensemble, sachant qu'une règle R_2 est dépendante d'une règle R_1 si l'application de R_1 peut "déclencher" l'application de R_2, c'est-à-dire s'il existe un SG G sur lequel l'application de R_1 produit un SG dans lequel peut être établie une projection de l'hypothèse de R_2 qui ne peut l'être dans G. Un algorithme est proposé pour déterminer si un ensemble de règles respecte cette condition. En plus de cette méthode, il existe une procédure simple pour reconnaître un autre type de règles à dérivation finie en s'assurant que leur syntaxe respecte certaines conditions : les règles "range-restricted".

Règles "range-restricted"

Une règle est dite "range-restricted" lorsque sa conclusion ne présente aucun sommet concept générique. Le terme est similaire à celui utilisé pour désigner les règles Datalog dont toutes les variables de la tête doivent apparaître dans le corps. Dans la figure 3.5, les deux règles R_1 et R_2 sont "range-restricted" alors que R_3 ne l'est pas.

Chaque règle R de ce type peut se fragmenter en un ensemble $\mathcal{D}(R)$ de règles dont l'hypothèse est identique à R et dont la conclusion ne présente qu'un seul sommet relation ou concept individuel provenant de la conclusion de R. Plus précisément, pour chaque sommet s de la conclusion de R, il existe une règle R' dans $\mathcal{D}(R)$ telle que :

- si s est un sommet concept individuel, alors l'hypothèse de R' est celle de R et la conclusion de R' est celle de R restreinte au sommet s ;
- si s est un sommet relation, alors l'hypothèse de R' est la somme disjointe de celle de R avec les sommets individuels de la conclusion de R ; et la conclusion de R' est restreinte au sommet s tel qu'il ait les mêmes voisins que dans R ;

Soit $\mathcal{R}$ un ensemble de règles "range-restricted" et F un SG. Un SG Q est déductible de F et des règles $\mathcal{R}$ ssi il est déductible de F et de leur décomposition $\mathcal{D}(\mathcal{R})$ (par contre, cette équivalence n'est plus valable si les contraintes - qui seront introduites en partie 3.3 - interviennent). Avec de telles règles, le problème de déduction dans une base de connaissances $K = (S, G, \mathcal{R})$ est NP-complet.

3.2.2 Sémantique logique.

La sémantique logique Φ s'étend de façon naturelle aux règles. La formule associée à une règle R est de la forme $\Phi(R) = \forall x_1 \ldots x_p \, ((hyp) \rightarrow \exists y_1 \ldots y_q \, (conc))$, où hyp et $conc$ sont les conjonctions d'atomes respectivement associées à l'hypothèse $R_{(0)}$ et à la conclusion $R_{(1)}$, $x_1 \ldots x_p$ sont les variables associées aux sommets de l'hypothèse et $y_1 \ldots y_q$

sont les variables associées aux sommets de la conclusion à l'exclusion des sommets frontières. Les mêmes variables sont associées aux sommets frontières se correspondant dans l'hypothèse et la conclusion (i.e. reliés par des liens de coréférence). Pour la règle R_2 de la figure 3.5, on obtient ainsi $\Phi(R_2) = \forall x \forall y \forall z\ (Bureau(x) \wedge Bureau(y) \wedge Bureau(z) \wedge adjacent(x,y) \wedge adjacent(y,z) \rightarrow Bureau(x) \wedge Bureau(z) \wedge proche(x,z))$ et pour la règle R_3 on obtient $\Phi(R_3) = \forall x(Chercheur(x) \rightarrow \exists(yProjet(y) \wedge member(x,y)))$. On remarque que les variables qui n'apparaissent que dans la conclusion sont existentiellement quantifiées.

Le mécanisme de chaînage avant associé à une base de connaissances $K = (S, F, \mathcal{R})$ est adéquat et complet par rapport à celui de la logique des prédicats. Étant donné un $SG\ Q$, $Q \geq K$ ssi $\phi(S), \phi(G), \phi(\mathcal{R}) \models \phi(Q)$. Ce résultat est établi si les graphes Q et G donnés et ceux résultant de l'application sur G des règles sont donnés ou mis sous forme normale.

Lorsque les règles de $\mathcal{R}$ sont "range-restricted", les formules issues de l'interprétation logique des règles de $\mathcal{D}(\mathcal{R})$ peuvent être mises sous la forme de clauses de Horn. En effet, chaque formule obtenue sera de la forme $\Phi(R) = \forall x_1\ ...\ x_p\ ((hyp) \rightarrow r(x_{i_1}, x_{i_2}, ..., x_{i_k}))$ (avec $1 \leq i_j \leq p$, pour $1 \leq j \leq k$ et k l'arité de r) ; ce qui peut se réécrire en une formule équivalente $\forall x_1\ ...\ x_p\ (hyp' \vee r(x_{i_1}, x_{i_2}, ..., x_{i_k}))$ où hyp' est une disjonction de littéraux négatifs équivalente à $\neg hyp$. L'expression $hyp' \vee r(x_{i_1}, x_{i_2}, ..., x_{i_k})$ est donc une disjonction composée de la série des littéraux négatifs apparaissant dans hyp' et d'un seul littéral positif (le prédicat r), ce qui correspond à une clause de Horn. Cette caractéristique des règles "range-restricted" garantit une déduction décidable.

3.3 Les contraintes de $\mathcal{SG}$.

Une contrainte de $\mathcal{SG}$ est un graphe bicoloré qui peut être qualifié soit de "négative", soit de "positive". Une contrainte *négative*, représente une connaissance interdite ou incohérente : l'un des sous-graphes monochrome de la contrainte est la partie *condition* et l'autre, la partie *interdiction*, représente un motif qui "ne doit pas être trouvé" si la partie *condition* l'a été au préalable. Les contraintes *positives* permettent de représenter des connaissances obligatoires. Elles sont composées de deux sous-graphes monochromes : la partie *condition* et la partie *obligation* ; et signifient que le motif *obligation* "doit être trouvé" si le motif *condition* l'a été au préalable.

La contrainte négative C_1 de la figure 3.8, par exemple, signifie que deux personnes qui travaillent ensemble ne peuvent partager le même bureau et la contrainte positive C_2 signifie que le bureau d'un directeur doit être situé près de celui d'une secrétaire.

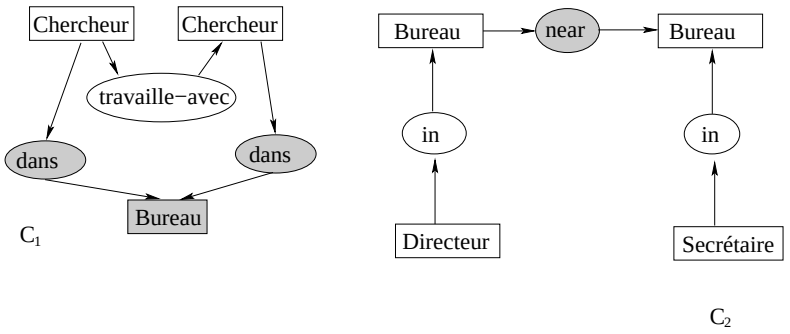


FIG. 3.8 – Exemple de contraintes négative et positive.

3.3.1 Définitions

Une contrainte positive C (resp. négative) est un SG bicoloré. $C_{(0)}$ est la partie *condition* de la contrainte et $C_{(1)}$ la partie *obligation* (resp. *interdiction*). Un SG G viole une contrainte positive (resp. négative) C s'il existe une projection de la condition de C dans la forme irredondante de G (resp. dans G) qui ne peut être étendue (resp. qui peut être étendue) à la projection du graphe C entier. Dans le cas contraire, on dit que G satisfait la contrainte C.

Par exemple, la contrainte négative C_1 de la figure 3.8 qui interdit à deux personnes travaillant ensemble de partager le même bureau est violée par le graphe G de la figure 3.1 qui représente un chercheur K qui partage le bureau d'un autre chercheur avec qui il travaille.

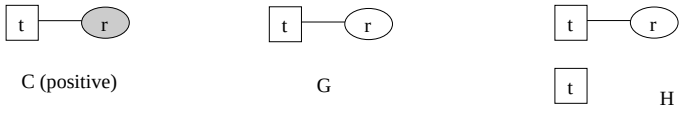


FIG. 3.9 – Violation de contrainte positive et redondance.

La figure 3.9 illustre la nécessité de mettre les graphes sous forme irredondante afin de vérifier s'ils respectent une contrainte positive donnée. Elle présente deux graphes équivalents G et H. Le premier satisfait la contrainte positive C et le second la viole alors qu'il devrait avoir un comportement identique. Pour éviter cette situation, il est nécessaire de mettre le graphe H sous forme irredondante (il est alors identique à G) afin qu'il satisfasse la contrainte C.

On dit que deux contraintes sont équivalentes ssi tout graphe qui viole l'une, viole aussi l'autre.

Toute contrainte négative C peut être mise sous la forme d'un SG monochrome. En d'autres termes, une contrainte négative bicolorée C est équivalente à la contrainte négative C' obtenue en attribuant aux sommets de $C_{(0)}$ la couleur 1. En effet, quand une contrainte négative C est violée, la projection de $C_{(0)}$ peut être étendue à la projection du SG C entier ; ce qui revient à entièrement projeter C. Or C est identique à $C'_{(1)}$, donc cela revient à projeter $C'_{(1)}$. Réciproquement, C' est violée ssi il est possible d'établir la projection π' de $C'_{(1)}$. Dans ce cas, le SG C peut aussi être projeté puisqu'il est à une couleur près identique à $C'_{(1)}$. Il est alors possible d'obtenir la projection π de $C_{(0)}$ en restreignant le domaine de π aux sommets de $C_{(0)}$ puisque ce dernier est un sous-graphe de C. π peut ensuite être étendue à C entier, puisque la projection π' a été établie, ce qui conduit à sa violation.

En plus de disposer d'une double définition (bicolorée et monochrome), une contrainte négative C peut être représentée par une contrainte positive C' équivalente. Cette dernière est obtenue en attribuant la couleur 0 à l'ensemble des sommets de C, et en ajoutant un sommet de couleur 1 dont le type (notons le $NotThere$) est incomparable à tous les autres et n'est utilisé dans aucun SG excepté les contraintes. Un SG viole la contrainte C ssi il viole C'. En effet, on a vu précédemment que si la projection de $C_{(0)}$ peut être étendue à la projection du SG C entier, on peut alors entièrement projeter C. Or C est identique à $C'_{(0)}$, donc dans ce cas $C'_{(0)}$ se projette aussi. Il faut maintenant vérifier que $C'_{(1)}$ est bien présent, ce qui conduit à la violation puisque le sommet de type $NotThere$ n'apparaît jamais dans les SG autres que les contraintes. Réciproquement, C' est violée ssi le SG $C'_{(0)}$ est projeté. On peut alors établir une projection π de C puisque C est identique à $C'_{(0)}$. $C_{(0)}$ étant un sous-graphe de C, on en obtient une projection en restreignant π aux sommets de $C_{(0)}$. Et cette projection peut être étendue à C puisque la projection π a été établie, ce qui conduit à sa violation.

La figure 3.10 présente la contrainte négative de la figure 3.8 sous les formes monochrome et positive. La contrainte négative C'_1 est obtenue en attribuant à tous les sommets de C_1 la couleur de la partie interdiction (grise) et la contrainte positive C''_1 l'est en coloriant C_1 avec la couleur de la partie *condition* (blanche) et en ajoutant le sommet $NotThere$ dans la partie *obligation*.

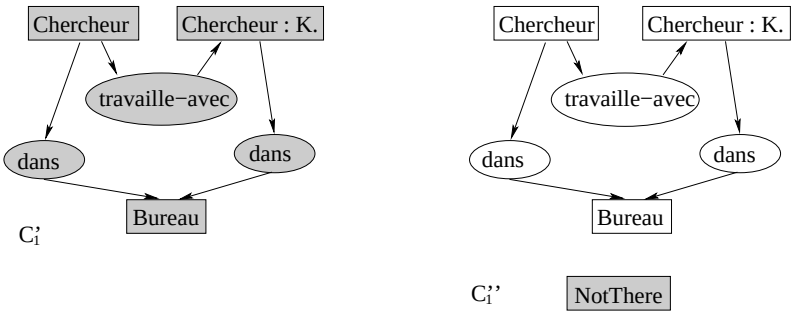


FIG. 3.10 – Contrainte négative sous forme monochrome et sous forme positive.

Le problème qui consiste à savoir si un graphe satisfait une contrainte positive donnée est $\prod_2^P$-complet. Si on restreint la forme de la contrainte afin qu'elle soit négative, le problème devient *co-NP*-complet.

G est dit cohérent relativement à un ensemble de contraintes $\mathcal{C}$ s'il satisfait toutes les contraintes de $\mathcal{C}$. Dans une base de connaissances $K = (S, G, \mathcal{C})$ - où $\mathcal{C}$ est un ensemble de contraintes négatives et positives, G un SG représentant la base d'assertions et S le support à partir duquel sont définis les graphes - l'interrogation de K n'est définie que si G est cohérent par rapport à $\mathcal{C}$ (on dit alors que K est cohérente). En d'autres termes, un SG Q considéré comme une requête ne peut être déduit de K que si K est cohérente et Q déductible de G.

3.3.2 Relations avec la logique

La déduction définie de la sorte pour une base de connaissances $K = (S, G, \mathcal{C})$ est non monotone. Tout ajout d'information à G peut violer une contrainte et rendre K incohérente. Comme rien ne peut être déduit d'une base de connaissances incohérente, les déductions précédentes ne sont plus valides. Cette notion de cohérence d'une base peut être traduite en logique des prédicats.

En ce qui concerne les contraintes négatives, elle s'obtient aisément en vérifiant qu'un SG G viole une contrainte négative $C = (C', \rho)$ ssi $\phi(S), \phi(G) \models \phi(C')$, ou C' est le SG monochrome qui définit la contrainte C, ρ la fonction de coloriage et $\phi(C')$ la formule logique associée à C'.

Pour modéliser en logique la vérification des contraintes positives, on introduit une substitution logique appelée *S-substitution*. Une *S-substitution* de $\phi(G)$ à $\phi(H)$ est une substitution σ des termes de $\phi(G)$ par les termes de $\phi(H)$ qui préserve les constantes de $\phi(G)$ et vérifie que pour tout atome $t(e_1, ..., e_k)$ de $\phi(G)$, il existe $t' \leq t$ tel que $t'(\sigma(e_1), ..., \sigma(e_k))$ soit un atome de $\phi(H)$.

Toute projection π d'un SG G dans un SG H définit une *S-substitution* σ de $\phi(G)$ à $\phi(H)$ et si H est sous forme normale la réciproque se vérifie aussi. A partir de cette propriété on peut dire qu'un SG G viole selon une projection π une contrainte C ssi la *S-substitution* σ de $\phi(C_{(0)})$ dans $\phi(G)$ associée avec π ne peut être étendue à une *S-substitution* de $\phi(C)$ dans $\phi(G)$.

3.4 Contraintes et règles

On considère maintenant une base $K = (S, F, \mathcal{R}, \mathcal{C})$ composée d'un support S, d'un ensemble de faits F, d'un ensemble de règles $\mathcal{R}$ et d'un ensemble de contraintes $\mathcal{C}$. Le mécanisme de déduction dans une telle base de connaissances est différent de celui associé à une base ne contenant soit que des contraintes, soit que des règles d'inférence. En effet, au cours d'une $\mathcal{R}$-dérivation sur F dans K, l'application d'une règle de $\mathcal{R}$ peut violer une contrainte positive, puis la connaissance manquante à l'origine de la violation peut être ajoutée par une nouvelle application de règle ; ce qui répare l'incohérence détectée.

La définition de la cohérence d'une base doit donc s'accommoder de cette particularité. $K = (S, F, \mathcal{R}, \mathcal{C})$ est considérée cohérente si pour tout graphe G incohérent obtenu par $\mathcal{R}$-dérivation à partir de F, il existe un SG H obtenu par $\mathcal{R}$-dérivation de G qui satisfait les contraintes $\mathcal{C}$.

Le problème d'interrogation n'étant défini que pour une base cohérente, un SG Q ne peut être déduit de K que si K est cohérente et Q déductible de $K' = (S, F, \mathcal{R})$. Dans le cas général, le problème n'est pas décidable mais lorsque les règles sont à dérivation finie, il le devient. En effet, pour de telles règles, la base est cohérente ssi $F^{\mathcal{R}}$ est cohérent (problème $\prod_2^P$-complet) et Q se déduit de K ssi il existe une projection de Q dans $F^{\mathcal{R}}$ (problème NP-complet).

3.4.1 Relations avec la logique

Pour interpréter en logique la notion de cohérence définie pour une base $K = (S, F, \mathcal{R}, \mathcal{C})$, il faut considérer les contraintes sous la forme de règles et les traduire avec ϕ. S'il existe un SG F' tel que $\phi(S), \phi(F), \phi(\mathcal{R}), \phi(\mathcal{C}) \models \phi(F')$, et tel que l'on n'ait pas $\phi(S), \phi(F), \phi(\mathcal{R}) \models \phi(F')$ où $\phi(\mathcal{C})$ est la traduction des contraintes de $\mathcal{C}$ considérées comme des règles, alors K est incohérente. La réciproque est vraie si les règles de $\mathcal{R}$ sont à dérivation finie mais ne l'est pas dans le cas général. Pour la même raison, les problèmes qui consistent à savoir (1) si $K = (S, F, \mathcal{R}, \mathcal{C})$ est cohérente et (2) si un SG Q peut en être déduit, ne sont pas décidables, mais le deviennent si $\mathcal{R}$ est à dérivation finie.

3.5 Conclusion

Ce chapitre présentait les différents composants de la famille $\mathcal{SG}$, à savoir : le support, le SG, les règles et les contraintes négatives et positives ainsi que la *projection* opération fondamentale à partir de laquelle sont définies les autres opérations importantes du formalisme telles que l'application d'une règle ou la vérification d'une contrainte.

Le travail réalisé dans cette thèse est basé sur une base de connaissances $K = (S, F, \mathcal{R}, \mathcal{C})$ où $\mathcal{R}$ est un ensemble de règles à dérivation finie et $\mathcal{C}$ un ensemble contenant des contraintes négatives ou positives. F vérifie l'hypothèse du monde clos ce qui en pratique signifie que toute paire de sommets différents appartient à la relation dif ; cette condition permet de préserver la complétude avec la logique classique (munie de $\neq$) de toute projection dans F d'un graphe source présentant un lien différence.

Chapitre 4

Les transformations

Un des principaux objectifs du web sémantique est de favoriser l'échange et l'exploitation de connaissances qui peuvent être représentées dans différents langages. Cela conduit à des problèmes d'*interopérabilité* entre langages : lorsqu'un outil de gestion des connaissances doit intégrer des connaissances décrites dans un langage différent de celui qu'il utilise pour représenter les siennes, il peut être nécessaire de définir une transformation du premier langage dans le second.

Dans le cadre du travail présenté ici, il est nécessaire d'établir une transformation du langage des TM vers celui des graphes conceptuels (et plus précisément la famille $\mathcal{SG}$) afin d'*opérationaliser* les connaissances décrites dans l'outil de gestion des connaissances ITM et d'améliorer les services proposés par ITM en le munissant de capacités de raisonnement.

Un certain nombre de transformations de ce type ont été proposées entre les TM et le langage RDF mais aucune n'a été réalisée entre les TM et le formalisme des graphes conceptuels. Cette partie présente trois transformations établies du langage des TM vers la famille $\mathcal{SG}$. Parmi ces trois transformations, une seule a été retenue car elle présentait les meilleures propriétés pour *opérationaliser* les connaissances d'ITM. La dernière section de ce chapitre est consacrée aux transformations existantes du langage des TM vers le langage RDF.

4.1 Transformations entre $XTMG_{MDK}$ et la famille $\mathcal{SG}$

L'ensemble de la connaissance d'ITM est représentée en TM[1], quel que soit le niveau (méta, modèle ou instance) auquel elle appartient. Cette homogénéité dans la représentation permet de développer trois types de démarches pour transformer une TM en un

[1]Dans cette partie, le terme TM désigne selon le contexte soit le formalisme $XTMG_{MDK}$, soit un graphe de ce formalisme.

SG. Ces dernières sont présentées dans la figure 4.1.

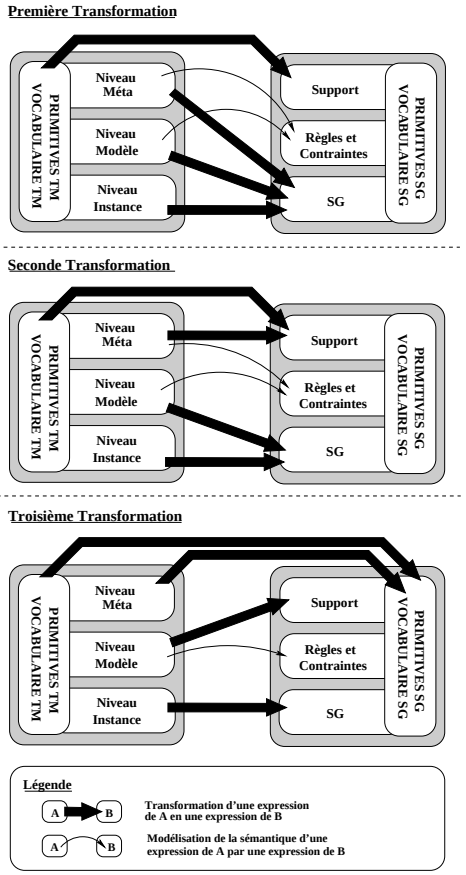


FIG. 4.1 – Trois transformations possibles entre le langage TM et la famille $\mathcal{SG}$.

- la première possibilité consiste à représenter les primitives et le vocabulaire TM (topic, association, etc) par des types du support, les connaissances TM des trois niveaux présentés à la figure 4.2 en une base de faits SG et la sémantique des connaissances des niveaux *méta* et *modèle* en des règles et contraintes ;
- la seconde à représenter les primitives et le vocabulaire TM ainsi que les connaissances du niveau *méta* par des types du support ; les connaissances TM des niveaux *modèle* et *instances* (voir fig. 4.2) l'étant par un SG. Comme dans la première transformation, la sémantique des connaissances définies dans les deux plus hauts niveaux est modélisée par des règles et des contraintes.
- La dernière consiste à transposer le niveau *méta* sur les constituants et opérations

de base du formalisme SG (il n'est donc pas représenté dans la famille $\mathcal{SG}$), tandis que les niveaux *modèle* et *instance* sont respectivement représentés par un support et un SG. La sémantique des connaissances définies au niveau *modèle* est modélisée par des règles et des contraintes.

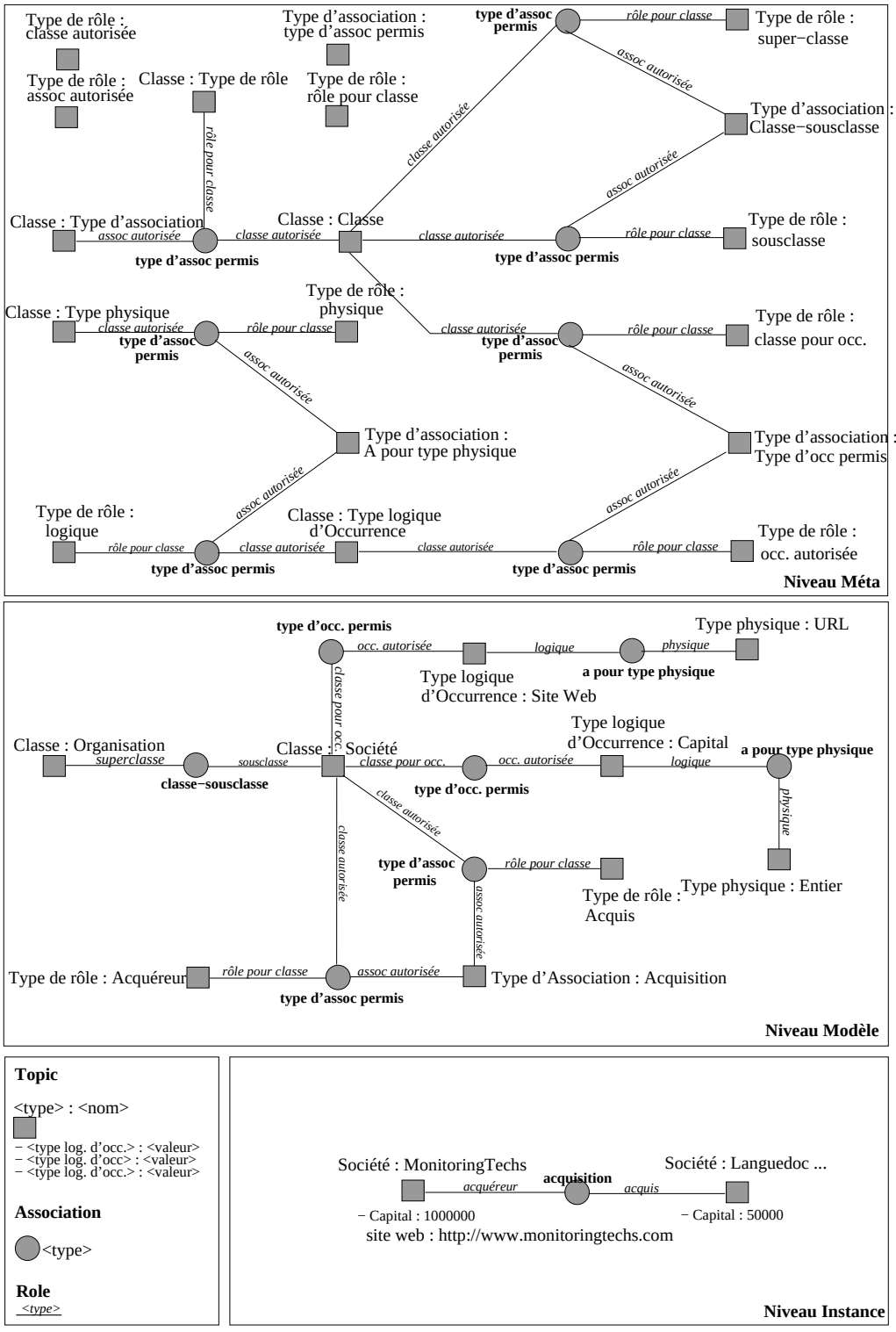


FIG. 4.2 – Un extrait de base ITM avec son niveau méta, modèle et instance.

4.1.1 Première transformation

Le support défini pour cette transformation représente les éléments du vocabulaire TM, à savoir : `Topic`, `Association`, `rôle`, `Occurrence` et `Valeur`, par des types de concepts T, A, R, O et V. Il ne compte qu'un type de relation `lien` qui représente le lien de dépendance qu'il peut exister entre deux éléments du vocabulaire TM lorsque la présence du premier dépend de celle du second. Il en est ainsi pour la valeur et son occurrence, l'occurrence et son topic, le rôle et son association, et enfin le topic qui participe à une association et son rôle.

Le SG image d'une telle transformation représente des connaissances provenant aussi bien du niveau méta, du niveau modèle que du niveau instance. Par exemple, la TM définie au niveau méta et présentée à la figure 4.3 représente la définition réflexive de la classe `Classe` par une association `Classe-instance`.

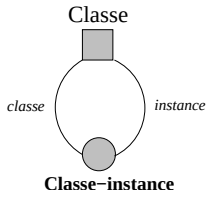


FIG. 4.3 – Définition au niveau méta du topic `Classe` (l'association *classe-instance* est exceptionnellement représentée dans la figure).

Le topic `Classe` peut être représenté en SG par le graphe de la figure 4.4. Dans ce graphe, les sommets concepts de type `T` représentent des topics, ceux de type `A` des associations et ceux de type `R` des rôles. Dans la suite, ils sont respectivement désignés par les termes *concept-topic*, *concept-association* et *concept-rôle*. La relation `lien` est représentée par un petit disque noir (pour une meilleure lisibilité). Le circuit à la périphérie du graphe représente l'instanciation réflexive du topic `Classe` (représenté par `[T : Classe]`) : cela signifie que ce topic est une instance de lui même. Ce circuit correspond à celui apparaissant dans la TM de la figure 4.3.

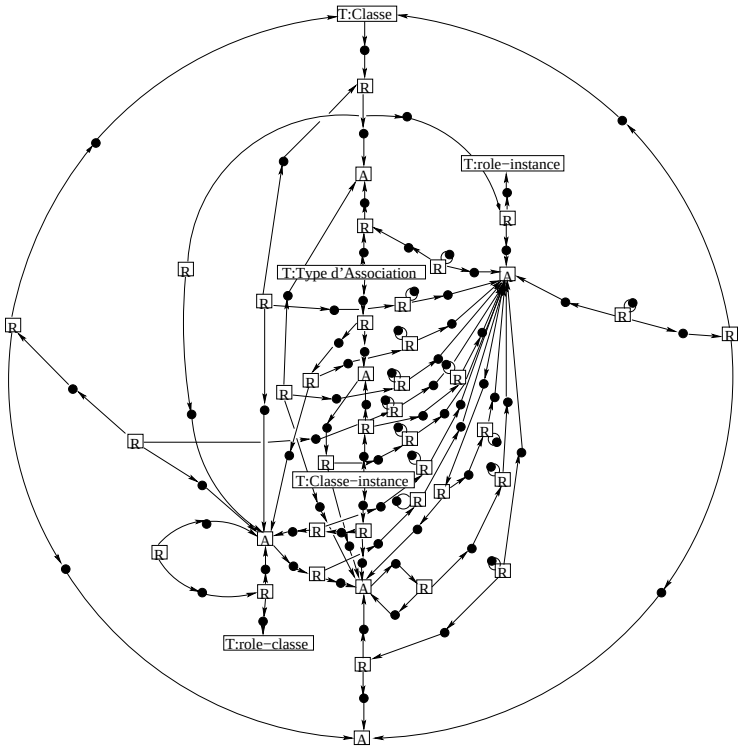


FIG. 4.4 – Le SG représentant la définition de la classe de topic `Classe`.

Les sommets disposés à l'intérieur du circuit représentent l'attribution du type `Classe-instance` à l'association qui instancie le topic `Classe`. Le type d'association `Classe-instance` est représenté par un sommet topic `[T : Classe-instance]` lui même instance d'un sommet topic `[T : Type d'Association]` représentant le type `Type d'association`. Toute instanciation d'un sommet s'effectue par l'intermédiaire d'un concept-association instance d'un concept-topic `[T : Classe-instance]` et dont les deux concept-rôles instances de `[T : role-instance]` ou `[T : role-classe]` permettent de distinguer les concept-topics instance de celui qui représente leur classe.

Cet exemple fait apparaître la particularité de l'approche : Le support défini est d'une pauvreté délibérée afin d'éviter à la représentation de dépendre de son pouvoir d'expression. La démarche consiste ensuite à associer à chaque TM un graphe suffisamment spécifique et discriminant pour éviter qu'il ne représente à lui seul deux TM différentes (c'est la forme du graphe qui est discriminante et non le vocabulaire utilisé). Considérons par exemple le graphe de la figure 4.5. Ce graphe ne peut être choisi pour représenter une association de type `Classe-instance`, puisqu'il n'est pas assez discriminant et à lui seul

il représente visiblement les associations de type quelconque établies entre une association et le topic `Classe-instance`. Pour construire le sous-graphe de la figure 4.4 associé à l'association de type `Classe-instance`, on a considéré que les topics ou associations ont toujours un type qui leur est attribué par une association elle-même toujours typée. Si le graphe présente un topic ou une association non typé, on le complète en ajoutant une association pour typer l'élément non typé. Cette association ne portant elle-même pas de type, elle déclenche une nouvelle fois la procédure de complétion précédente ; et ainsi de suite récursivement. Or, l'ajout récursif d'associations typées qui attribuent un type peut se poursuivre indéfiniment si on ne choisit pas un motif final[2] représentant le caractère réflexif de la définition de ces associations. Pour mettre un terme au procédé récursif de complétion, on ajoute au graphe un motif représentant une association s'attribuant elle-même son type.

FIG. 4.5 – Un graphe SG trop simple pour représenter une seule TM de façon unique.

Au cours de cette transformation, les propriétés sémantiques de certaines TM sont modélisées dans la famille $\mathcal{SG}$ en utilisant des règles ou des contraintes d'une taille et complexité similaires à celles des SG obtenus par la procédure de construction précédente. Cette partie de la transformation est détaillée en annexe, à la section A.4.

D'une manière générale, cette transformation associe à chaque primitive TM du niveau méta un motif discriminant et un ensemble de règles qui lui sont spécifiques. Cette transformation offre l'avantage (1) de représenter les primitives tout en tenant compte des éventuelles évolutions du niveau méta en fournissant sans perte d'informations un motif correspondant à toute nouvelle primitive, (2) de modéliser par des règles et contraintes les propriétés sémantiques de ces primitives. Cependant, la généricité de la méthode a des conséquences négatives sur la lisibilité des représentations et les performances de la projection. En effet, le nombre important de sommets produits en $\mathcal{SG}$ pour une TM ralentit les opérations et rend la lecture difficile.

4.1.2 Seconde transformation

[2]Il faut noter qu'il existe plusieurs motifs terminaux différents.

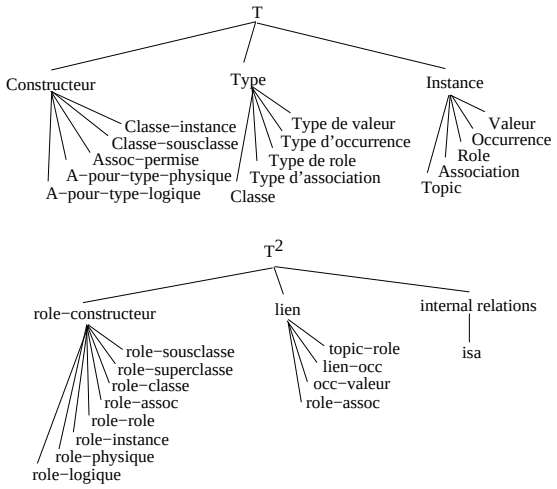


FIG. 4.6 – Un support représentant le niveau méta permettant d'établir la seconde transformation.

La seconde démarche consiste à exploiter davantage le pouvoir d'expression du support afin de réduire la taille des représentations. Les constructeurs du méta-modèle ITM sont traduits en types dans le support et les niveaux inférieurs sont transformés en une base de faits SG. Cette transformation est partielle ; car certaines TM n'ont aucune image.

La hiérarchie des types de concepts se divise en trois parties, chacune étant associée au niveau méta, modèle ou instance d'ITM. La figure 4.6 présente un tel exemple de support.

Les primitives sont représentés dans le support sous le type de concept `Constructeurs` et non plus au niveau assertionnel par un graphe. Chaque rôle associé à une primitive est représenté par un type de relation binaire. Par exemple, l'instanciation et le sous-classement sont définis dans le support mais sont représentés au niveau assertionnel et les propriétés des associations qui leur sont associées sont modélisées par des règles et des contraintes SG. Le vocabulaire permettant de décrire le niveau modèle est introduit dans le support sous le type `Type` et celui permettant de décrire le niveau instance sous le type `Instance`.

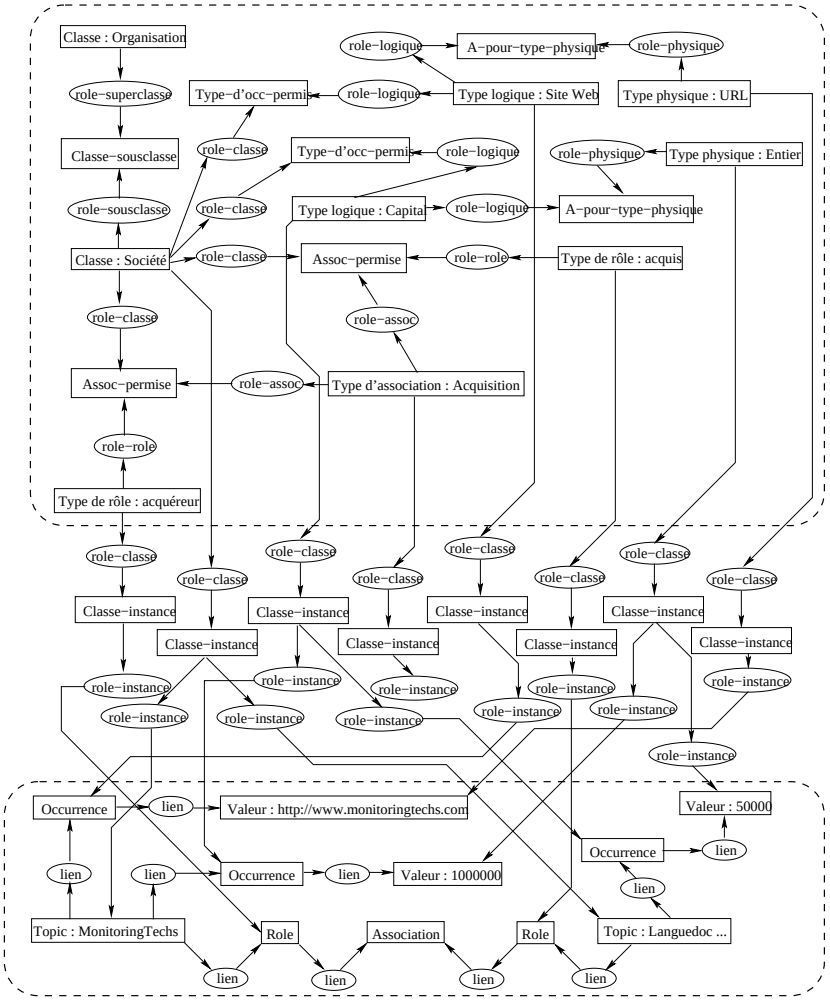


FIG. 4.7 – Un SG représentant des connaissances des niveaux modèle et instance.

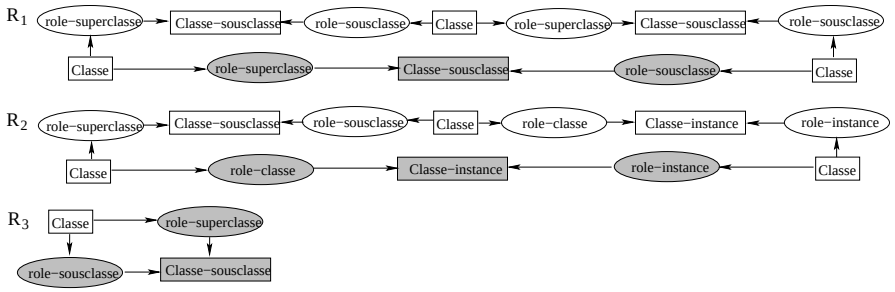


FIG. 4.8 – Règles SG représentant les propriétés du lien de sous-classement.

En raison d'une légère perte d'information, cette transformation du langage TM vers la famille $\mathcal{SG}$ n'est pas totale. En effet, il n'y a pas d'équivalent $\mathcal{SG}$ pour les définitions réflexives des constructeurs mais cela n'a que peu de conséquences en pratique, car les ajouts ou modifications au niveau méta sont rares.

Au niveau assertionnel, le SG représente des connaissances provenant à la fois des niveaux modèle et instance (la figure 4.7 en illustre un exemple). Le sous-graphe qui correspond à la TM du niveau instance (celui entouré en bas de figure) est celui induit par les sommets de G qui ne sont pas reliés à un sommet de type `Classe-instance` par une relation de type `rôle-classe`. Celui qui rassemble le reste des sommets correspond au niveau modèle (en haut de la figure). Le topic `MonitoringTechs` est représenté dans le premier sous-graphe sous la forme d'un sommet concept `[Topic : MonitoringTechs]`, tandis que sa classe `Société` est représentée sous la forme d'un sommet concept `[Classe : Société]` dans le second.

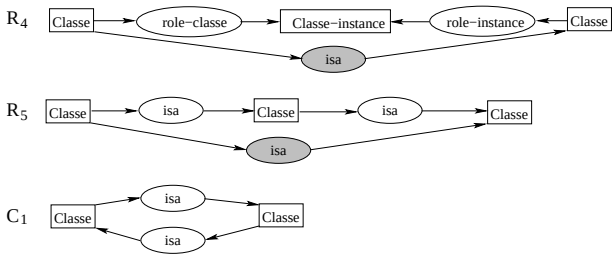


FIG. 4.9 – Contrainte et règles SG permettant de préserver acircuitique le lien d'instanciation.

Entre ces deux sous-graphes apparaissent des sommets de type `Classe-instance` qui représentent l'attribution d'un type du niveau modèle à un élément du niveau instance.

Par exemple, le topic `MonitoringTechs` de classe `Société` se représente par un sommet `[Classe-instance]` relié par deux relations `rôle-classe` et `rôle-instance` aux sommets `[Classe : Société]` et `[Topic : MonitoringTechs]`.

Les figures 4.8 et 4.9 présentent les règles et contraintes modélisant les propriétés de l'instanciation et du sous-classement. Les règles R_1 et R_3 spécifient la transitivité et réflexivité du sous-classement représenté par le type `Classe-sousclasse`. La règle R_2 précise que toute instance d'une classe est instance de sa super-classe. La taille de la règle R_1 est bien inférieure à celle de la même règle établie lors de la transformation précédente (figure A.6). La contrainte C_1 spécifie l'irréflexivité de la relation `isa`. Cette contrainte jumelée aux règles R_4 et R_5 permet de préserver acircuitique le réseau induit par le lien d'instanciation (i.e. le sous-graphe induit par les sommets de type `Classe-instance`, `rôle-instance` et `rôle-classe`).

Comparé à la première approche, le caractère discriminant dans la représentation d'un constructeur n'est plus la forme d'un graphe mais l'identifiant du type qui lui correspond dans le support ; ce qui réduit la taille des représentations, les rend plus lisibles et améliore les performances de la projection. Cependant des règles et contraintes coûteuses sont toujours utilisées pour prendre en compte la sémantique des primitives et bien que cette approche offre comme la précédente une certaine généricité, il en résulte des temps d'inférence et de vérification de contraintes insatisfaisants. La définition de certaines règles ou contraintes pourrait être évitée en exploitant davantage les capacités du support. Par exemple, la sémantique des notions d'instanciation et de sous-classement formalisée ici par des règles et contraintes pourrait être prise en charge par le support qui fournit ses propres moyens de sous-classement et d'instanciation (la généricité de la représentation s'en trouvant restreinte).

4.1.3 Troisième transformation

C'est précisément l'objectif de la troisième approche. Cette transformation consiste à transposer les propriétés importantes du niveau *méta* sur les primitives intrinsèques à la famille $\mathcal{SG}$, puis à associer un support au niveau *modèle* et un SG au niveau *instance*.

La figure 4.10 présente une partie du support associé au niveau modèle de la figure 4.2. Il représente les classes, les types d'association et les types de valeurs du niveau modèle dans la hiérarchie des types de concepts dont l'ordre est celui induit par le sous-classement en TM. Les types de rôles et d'occurrences sont représentés dans la hiérarchie des types de relations. Comme la notion de sous-classement en TM ne s'applique pas à ces types, ils sont disposés en râteau sous un type fédérateur.

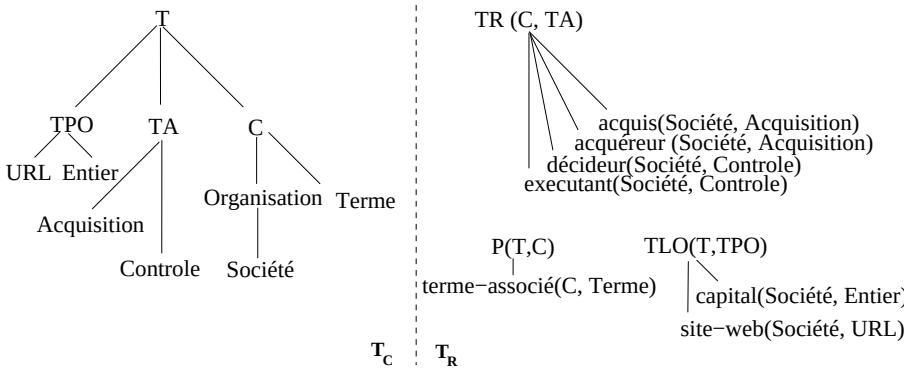


FIG. 4.10 – Un extrait du support associé au niveau modèle d'ITM.

Le SG correspondant au niveau instances est présenté à la figure 4.11. Il représente les sociétés de la TM et l'association d'acquisition à laquelle elles participent par des sommets concepts individuels et un sommet générique. Les rôles et les occurrences prennent en SG la forme de relations. Il faut remarquer qu'en TM le nombre des topics jouant un rôle dans une association d'un certain type peut être variable [3]. Afin de préserver cette particularité, les associations sont représentées en $\mathcal{SG}$ sous la forme de sommets concepts et non pas de sommets relation, car un sommet concept peut avoir un nombre non fixé de relations voisines alors qu'une relation a une arité fixe.

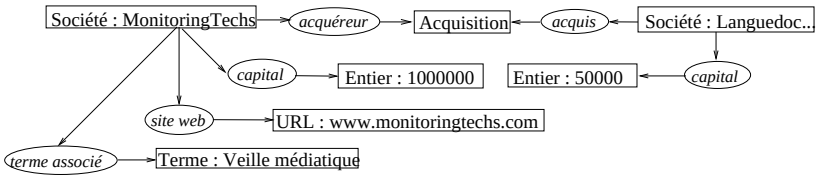


FIG. 4.11 – Le SG obtenu à partir de la TM du niveau instance de la figure 2.22.

Dans cette approche, un nombre significativement inférieur de sommets est généré par rapport aux démarches précédentes. Pour une même TM, le nombre de sommets n_i est toujours significativement[4] supérieur à celui n_j respectivement générés au cours des $i^{eme} > j^{eme}$ démarches présentées (ceci implique que $n_i/n_j > 1$). Les performances

[3] A condition que la cardinalité minimale éventuellement définie au niveau modèle pour le type de l'association soit strictement inférieure à la cardinalité maximale

[4] Si on considère deux couples de graphes G_i, H_i et G_j, H_j, respectivement obtenus pour cette même TM au cours des $i^{eme} > j^{eme}$ démarches, la complexité (établie au pire des cas) d'une projection de G_i sur H_i entretient un rapport exponentiel constant avec celle d'une projection de G_j sur H_j. Plus

de l'opération de projection s'en trouvent donc améliorées et il en est de même pour la lisibilité.

Une autre caractéristique de cette transformation consiste à directement prendre en compte les propriétés sémantiques des primitives par le formalisme $\mathcal{SG}$. Ceci améliore aussi les performances de la déduction car aucune règle ne modélise la sémantique des associations de sous-classement et d'instanciation comme dans les deux démarches précédentes. La définition d'autres règles et contraintes associées aux types du niveau modèle reste pour autant permise. Elles permettent par exemple d'exprimer la transitivité d'un certain type d'associations. La figure 4.12 présente une règle R et une contrainte négative C^- formalisant la nature hiérarchique d'une association de type `Contrôle` et de nature `Fait`. La règle jumelée à la contrainte évite à de telles associations de former un circuit.

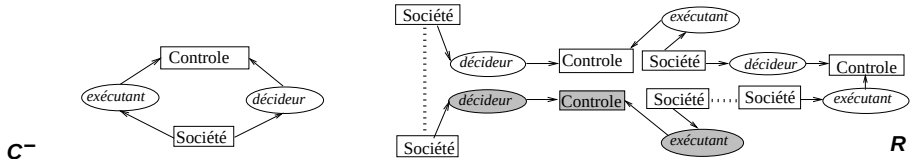


FIG. 4.12 – Une règle et une contrainte négative obtenues à partir de f_{H2RC^-}.

Cette démarche ne peut cependant pas être menée sur l'ensemble des informations représentées dans ITM car certaines n'ont pas vocation à être considérées comme des connaissances. En plus des connaissances d'ordre assertionnel (annotations), ITM regroupe des informations terminologiques (thésaurus) ou structurelles (organisation de document, de dossiers) dont certaines relèvent davantage de la donnée brute et sont utilisées par les traitements logiciels de l'application. La démarche retenue pour transformer une TM en SG consiste donc à restreindre le champ d'application de la troisième approche sur la portion d'ITM dédiée aux annotations. La partie suivante définit en détail la transformation.

4.2 Description détaillée de la troisième transformation

La transformation de la TM du niveau modèle vers le famille $\mathcal{SG}$ est réalisée par quatre fonctions. La fonction f_{TM2S} permet d'obtenir un support à partir du niveau modèle d'ITM (les exemples sont relatifs à la figure 4.2) ; tandis que les fonctions f_{MAX2C^-} et f_{MIN2C^+} traduisent les cardinalités maximum et minimum de l'ontologie, respectivement en contraintes négatives et positives ; et la fonction f_{H2RC^-} transforme la nature

précisément, si la complexité d'une projection de G_j dans H_j est en $o(k^{n_j})$ et celle de G_i dans H_i est en $o(k^{n_i})$, alors $o(k^{n_j}) < o(k^{n_i})$ car $k^{n_i} = (k^{n_j})^{(n_i/n_j)}$ et $n_i/n_j > 1$.

hiérarchique d'un type d'association de l'ontologie en une règle et une contrainte négative.

Deux transformations f_{TM2SG} et f_{SG2TM} ont été définies entre le niveau instances et la famille $\mathcal{SG}$. La première permet de traduire une TM de l'espace des annotations respectant le niveau modèle en un SG respectant le support S obtenu par application de f_{TM2S} sur la TM tm_{modele} du niveau modèle. La seconde réalise l'opération inverse ; à savoir la traduction d'un SG défini à partir du support S en une TM respectant le niveau modèle.

Les transformations associées au niveau modèle et au niveau instances ne sont définies que pour les TM respectant les contraintes introduites au chapitre `Les Topic Maps` dans les sous-sections 2.4.2 et 2.4.3. Les définitions incomplètes au niveau modèle ou instances sont donc ignorées.

4.2.1 Obtention d'un support à partir de l'application de f_{TM2S} sur une TM modèle

Obtention de la hiérarchie des types de concept

Trois types de concepts C, TA et TPO sont créés comme sous-type d'un type universel $\top$. Ces sous-types représentent respectivement le supertype des classes de topic, types d'association et types physiques d'occurrence. Tous les topics t dont le *type* est `classe`, `type d'association` ou `type physique` deviennent respectivement par f_{TM2S} des types de concepts sous-types de C, TA et TPO et sont identifiés par $nom(t)$[5]. L'ordre partiel sur T_C est induit par celui de l'association `classe-sousclasse`. Par exemple, les topics `Société` et `Organisation` (du niveau modèle) deviennent par f_{TM2S} des éléments de T_C ordonnés de la manière suivante $Société \leq Organisation \leq C$.

[5]Rappel : la fonction nom associe à chaque topic une sorte d'URI appelé le *PSI* qui identifie le topic dans la TM.

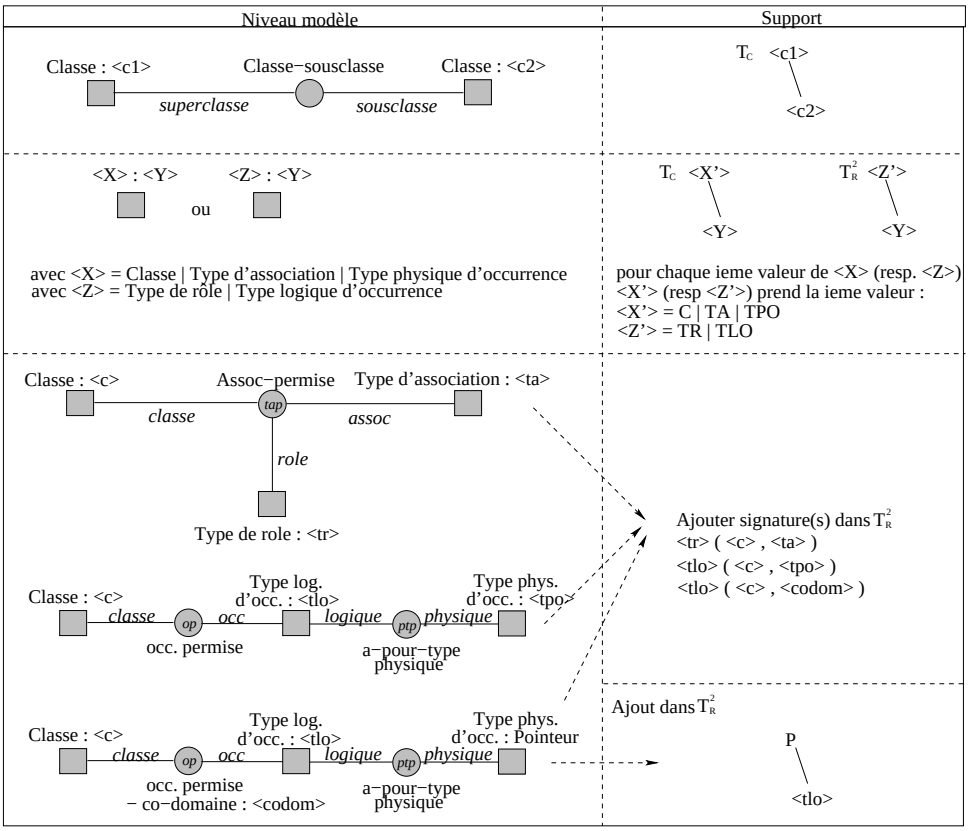


FIG. 4.13 – Traduction par f_{TM2S} de la TM du niveau modèle en un support.

Obtention de la hiérarchie des types de relation

Trois types de relations TR, TLO et P ayant respectivement pour signature $\sigma(TR) = (C, TA)$, $\sigma(TLO) = (\top, TPO)$, $\sigma(P) = (\top, C)$ sont créées pour représenter respectivement les supertypes des rôles, des types logiques d'occurrence et les types de pointeurs.

La figure 4.13 présente la transformation établie par f_{TM2S} d'un sous-graphe TM du niveau modèle en un type du support. La figure 4.10 présente les hiérarchies de types de concepts et relation correspondant à une partie du résultat de l'application de la transformation f_{TM2S} sur la TM du niveau modèle de la figure 4.2.

Tous les topics *tr* dont le *type* est `type de rôle` qui participent à une association *tap* ternaire de *type* `type d'association permis` avec des topics *c* de *type* `classe` et *ta* de *type* `type d'association` deviennent par f_{TM2S} des relations identifiées par $nom(tr)$ sous-relations de TR et ont pour signature $\sigma(nom(tr)) = (f_{TM2S}(c), f_{TM2S}(ta))$.

Par exemple, le topic (du niveau modèle) `acquis` relié par `type d'association permis` à `Société` et `Acquisition` devient par f_{TM2S} une relation *acquis* mise sous TR et ayant pour signature *σ(acquis)=(Société, Acquisition)*.

Tous les topics *tlo* dont le *type* est `type logique d'occurrence` sont reliés par une association *ptp* de *type* `a pour type physique` à un topic *tpo* et par une association *op* de *type* `type d'occurrence permis` à un topic classe *c*. Ceux pour lesquels *tpo* est différent de `pointeur` deviennent par f_{TM2S} des relations identifiées par $nom(tlo)$ sous-relations de TLO et ont pour signature $\sigma(nom(tlo)) = (f_{TM2S}(c), f_{TM2S}(tpo))$. Pour ceux dont *tpo* est égal à `pointeur`, si *op* possède une occurrence de valeur nom_{codom} et de type logique `co-domaine`, ils engendrent par f_{TM2S} des relations $nom(tlo)$ sous-relations de P et de signature $\sigma(nom(tlo)) = (f_{TM2S}(x), f_{TM2S}(t_{codom}))$, avec t_{codom} le topic de nom nom_{codom} ; par contre si *op* ne possède pas d'occurrence de type logique `co-domaine` la signature de ces relations est $\sigma(nom(tlo)) = (f_{TM2S}(x), C)$.

4.2.2 Obtention d'un ensemble de contraintes à partir de l'application de f_{TM2C^-} et f_{TM2C^+} sur une TM modèle

Si l'association *tap* précédemment introduite (rassemblant un topic rôle tr, un topic classe c et un topic association ta) possède une occurrence (m, *tpo*,`cardinalité maximum`) et de valeur m (cf. fig 4.14), on lui associe par f_{MAX2C^-} une contrainte négative SG G représentant $m + 1$ concepts génériques de type $f_{TM2S}(c)$ chacun d'entre eux reliés aux autres par un lien de "différence" et par une relation de type $f_{TM2S}(tr)$ à un unique sommet concept générique de type $f_{TM2S}(ta)$.

De la même façon, si *tap* possède une `cardinalité minimum` m, $f_{MIN2C^+}(tap)$ produit une contrainte positive SG dont la partie *obligation* est identique à G, à ceci près qu'elle comporte m sommets concepts génériques de type $f_{TM2S}(c)$ et le même nombre de sommets relations de type $f_{TM2S}(tr)$. La partie *condition* ne contient que le sommet de type $f_{TM2S}(ta)$ en tant que sommet frontière.

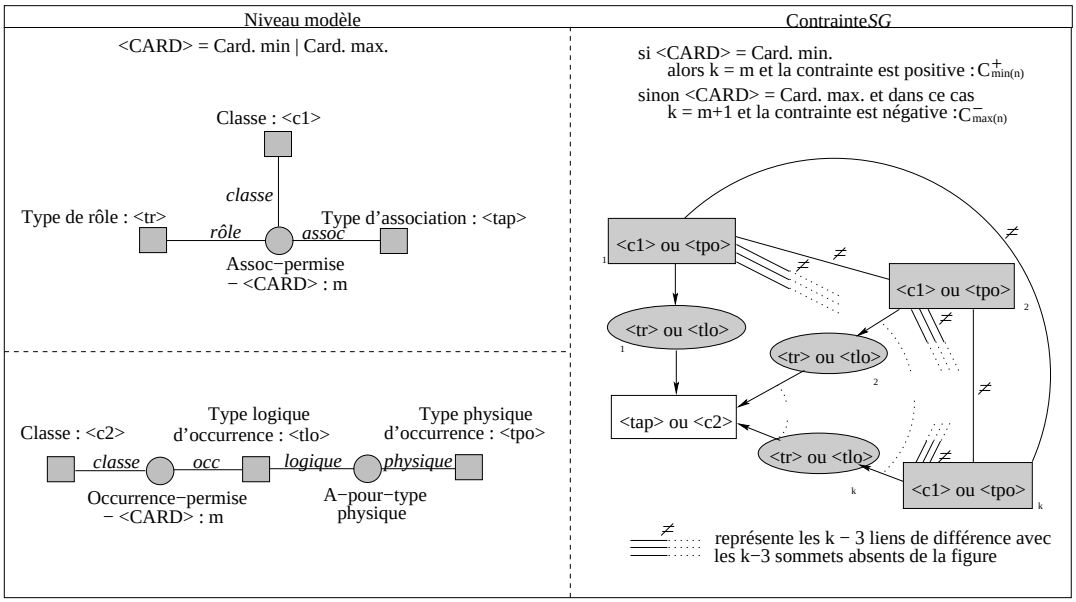


FIG. 4.14 – Transformation des cardinalités en contraintes.

Si l'association *op* introduite dans la partie précédente (rassemblant un topic classe c, un topic type logique *tlo* lui même relié à un topic type physique *tpo*) possède une cardinalité maximum (resp. cardinalité minimum) de valeur m (cf. fig 4.14), on produit par f_{MAX2C^-} (resp. par f_{MIN2C^+}) une contrainte négative SG (resp. une contrainte positive SG) selon laquelle un sommet concept de type $f_{TM2S}(c)$ ne doit pas être (resp. doit être) relié à $m+1$ (resp. m) sommets concepts génériques différents de type $f_{TM2S}(tpo)$ par $m+1$ (resp. m) relations de type $f_{TM2S}(tlo)$.

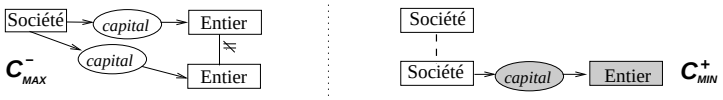


FIG. 4.15 – Une contrainte négative interdisant à une société d'avoir plus d'un capital et une contrainte positive l'astreignant à en avoir au moins un.

La figure 4.15 montre les contraintes négative et positive obtenues par les transformations f_{MAX2C^-} et f_{MIN2C^+} sur une cardinalité maximum et minimum signifiant dans ITM que toute instance de la classe `Société` a une et une seule occurrence de valeur entière et de type logique `Capital`.

4.2.3 Obtention d'un ensemble de règles et contraintes à partir d'une TM modèle

On considère à présent les types d'`association orientée` t_{ao} qui sont respectivement reliés par deux associations `type d'association orientée permis` à un topic t_{rp} et t_{rs} jouant les rôles de type `prédécesseur` et `successeur`, ainsi qu'à deux topics c_p et c_s représentant les classes de topics t qui peuvent respectivement jouer les rôles de type t_{rp} et t_{rs} dans une association de type t_{ao} au niveau instance.

Le topic t_{ao} peut porter une occurrence de type `hiérarchie` dont la valeur indique la forme du réseau que doivent dessiner les associations de type t_{ao}. Les valeurs prises en compte dans cette transformation sont `graphe sans circuit transitif` et `association transitive` (la valeur par défaut est `graphe sans circuit transitif`). La forme des règles et des contraintes dépend de la valeur de cette occurrence.

Dans la suite, si $f_{TM2S}(c_p)$ et $f_{TM2S}(c_s)$ sont comparables, x désigne celui qui est super-type de l'autre lui-même représenté par y, sinon ils désignent chacun un type différent de l'autre.

En général, c_p et c_s sont comparables. Dans ce cas on est confronté à trois possibilités :

- Dans le cas où c_p est super-classe de c_s, tout topic successeur dans un association orientée peut être prédécesseur dans une autre association orientée (l'inverse n'étant pas forcément permis).
- Dans le cas inverse où c_s est super-classe de c_p, tout topic prédécesseur dans une association orientée peut être successeur dans une autre association orientée (l'inverse n'étant pas forcément permis non plus).
- Dans le cas où c_s et c_p sont égaux, tout sommet prédécesseur ou successeur dans une association orientée peut être prédécesseur ou successeur dans une autre association orientée.

Le cas où c_p et c_s sont incomparables est rare mais néanmoins permis par l'ontologie. On considère dans l'ontologie un type d'association orientée t_{ao} pour lequel c_p et c_s sont incomparables. On définit ensuite l'ensemble $\mathcal{C}_{ao}$ de tous les chemins a_0, a_1 ... a_n formés par ces associations en les parcourant de telle sorte que a_{i+1} soit une association à laquelle participe en tant que prédécesseur un topic successeur de a_i. On est alors confronté à deux cas :

- Au niveau instances, on considère les associations de type t_{ao} auxquelles participent des topics dont le type est sous-classe soit de c_s soit de c_p mais pas des deux à la fois. Dans ce cas, la taille des chemins de $\mathcal{C}_{ao}$ est d'au plus un (par conséquent, un graphe sans circuit transitif défini par ces associations aura une hauteur d'au plus un). En effet, un topic successeur (resp. prédécesseur) d'une telle association orientée est de type c_s (resp. de type c_p). Il ne peut pas être prédécesseur (resp. successeur) dans une autre association orientée. S'il l'était il serait nécessairement de type c_p (resp. c_s). Or, un topic ne peut être à la fois de deux types incomparables

c_s et c_p que s'il est d'un type sous-classe de ces deux types (ce qu'on a rejeté au départ).

– Par contre, si les topics peuvent avoir pour type une sous-classe quelconque de c_p ou c_s (quelle leur soit commune ou pas) alors il n'y a pas de contrainte sur la taille des chemins de $\mathcal{C}_{ao}$; à ceci près que si le type d'un topic t participant à l'une des associations orientées n'est pas sous-classe du type c_p (resp. c_s) alors t ne peut être le prédécesseur (resp. le successeur) d'une autre association orientée.

Association transitive ou graphe sans circuit transitif

Lorsque l'occurrence `hiérarchie` prend la valeur `association transitive` ou `graphe sans circuit transitif`, f_{H2RC^-} associe à t_{ao} une règle R_1^H exprimant la transitivité de l'association t_{ao}. Dans le cas où la valeur est `graphe sans circuit transitif`, f_{H2RC^-} associe en plus une contrainte négative C_1^H interdisant à t_{ao} de former un circuit avec un topic. La figure 4.16 présente le schéma associé à une telle transformation.

C_1^H est formée de deux sommets concepts de type $f_{TM2S}(x)$ et $f_{TM2S}(t_{ao})$ et de deux relations de type $f_{TM2S}(tr_p)$ et $f_{TM2S}(tr_s)$ reliant le premier au second (et orientée dans ce sens).

L'hypothèse de la règle R_1^H est composée de trois concepts génériques s_x, s_y et s'_y de type $f_{TM2S}(x)$ pour le premier et $f_{TM2S}(y)$ pour les deux autres. Le premier et le second sont reliés à un concept générique s_a de type $f_{TM2S}(t_{ao})$ par deux relations de type $f_{TM2S}(t_{rp})$ et $f_{TM2S}(t_{rs})$. Il en est de même pour le second et le troisième. La conclusion est composée d'un sommet relation de type $f_{TM2S}(t_{ao})$ relié par $f_{TM2S}(t_{rp})$ et $f_{TM2S}(t_{rs})$ au premier et troisième sommet.

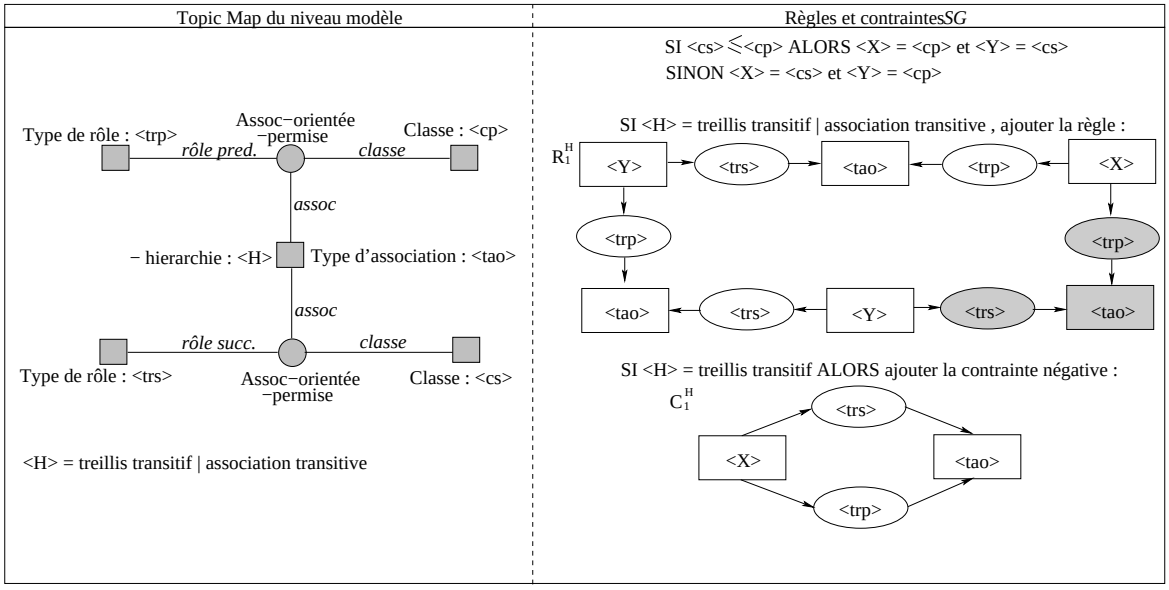


FIG. 4.16 – Transformation f_{H2RC^-} d'un type d'association orientée en règles et contraintes.

4.2.4 Obtention d'un SG à partir d'une TM annotation

La fonction f_{TM2SG} présentée sous forme graphique à la figure 4.17 permet à partir d'une TM annotation issue de la base ITM d'obtenir un SG à ajouter à la base de connaissances $\mathcal{SG}$ (les exemples sont relatifs à la TM du niveau instance de la figure 2.22 transformée en le SG de la figure 4.11).

Obtention des sommets concepts à partir des topics et associations

Tous les topics t dont le nom est spécifié sont transformés par f_{TM2SG} en un sommet concept individuel étiqueté $type(t) : nom(t)$; de plus $nom(t)$ est inséré dans l'ensemble des marqueurs individuels I. Lorsque nom n'est pas défini pour t, f_{TM2SG} engendre un sommet concept générique étiqueté $type(t)$.

Par exemple, le topic `MonitoringTechs` de *type* `Société` est transformé par f_{TM2SG} en le sommet concept individuel `[Société :MonitoringTechs]`.

Toute association a est transformée en un sommet concept générique étiqueté $type(a)$. Par exemple, l'association de *type* `acquisition` est transformée par f_{TM2SG} en le sommet concept générique `[Acquisition :*]`.

Obtention des sommets relations à partir des rôles et des occurrences

Chaque arête e reliant une association a à un topic t est transformée par f_{TM2SG} en un sommet relation d'étiquette $type(e)$ dont les deux arêtes incidentes étiquetées resp. 1 et 2 le relient resp. à $f_{TM2SG}(t)$ et $f_{TM2SG}(a)$.

Par exemple, l'arête portant le rôle `acquéreur` est transformée par f_{TM2SG} en le sommet relation `acquéreur` le reliant aux deux sommets concepts précédents.

Chaque occurrence $o = (v, tpo, tlo)$ d'un topic ou d'une association x engendre par f_{TM2SG} un sommet relation d'étiquette tlo dont les deux arêtes le relient resp. au sommet concept $f_{TM2SG}(x)$ et à un nouveau sommet concept individuel c_v étiqueté $[tpo : v]$.

Par exemple, l'occurrence `1000000` du topic `MonitoringTechs` de type logique `capital` et de type physique `Entier` devient par f_{TM2SG} un sommet concept d'étiquette `[Entier :1000000]` relié au sommet concept `[Société :MonitoringTechs]` par un sommet relation `capital`.

Si v est égale à $\varnothing$ (valeur indéfinie), le nouveau sommet concept c_v est générique et s'étiquette $[tpo : *]$.

Dans le cas où tpo est égal à `pointeur`, la seconde arête est reliée au sommet concept $f_{TM2SG}(y)$, avec y le topic dont $nom(y) = v$.

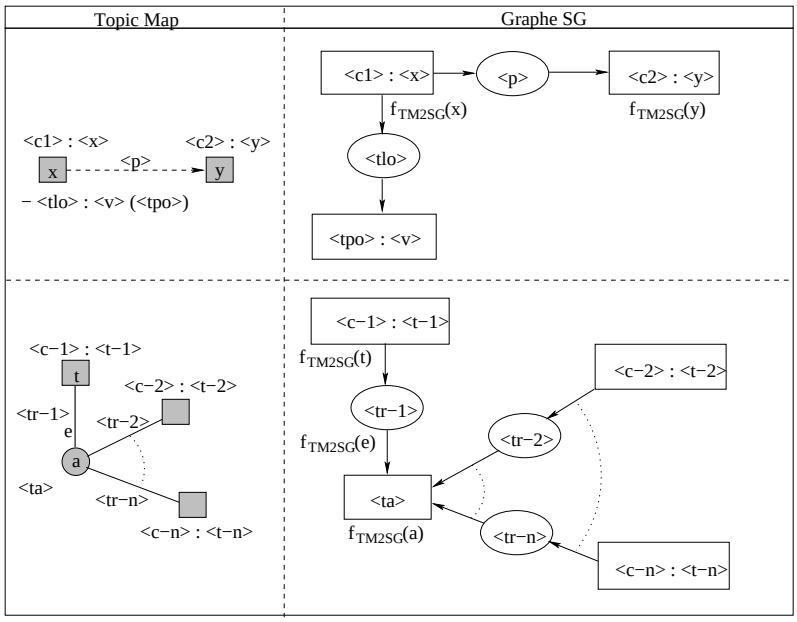


FIG. 4.17 – Transformation du niveau instance en SG de la base de faits.

4.2.5 Obtention d'une TM $XTMG_{MDK}$ à partir d'un SG

La transformation f_{TM2SG} est déterministe (c'est une fonction) et injective. Cela signifie d'une part qu'à chaque TM correspond au plus un SG et d'autre part que deux TM différentes possèdent deux SG images différents (lorsque les images existent). Cette propriété garantit l'existence d'au moins une fonction inverse injective de l'ensemble image de f_{TM2SG} vers l'ensemble des TM du domaine de f_{TM2SG}. Les parties suivantes définissent la fonction f_{SG2TM} qui est l'une des fonctions inverses de f_{TM2SG}. Elle permet d'obtenir une TM $tm = (T, A, E, type, nom)$ à partir d'un SG $G = (C_G, R_G, E_G, l_G)$ obtenu par f_{TM2SG}.

Obtention des topics et associations à partir des sommets concepts

Chaque sommet concept $c \in C_G$ de type $t \leq C$ et de marqueur individuel m est transformé par f_{SG2TM} en un topic x de T identifié par m ($nom(x) = m$) et dont le $type$ est $f^{-1}_{TM2S}(t)$[6].

Si $m = *$ et qu'il existe dans T un topic x pour lequel la fonction nom n'est pas définie tel que $f_{TM2SG}(x) = c$ alors $f_{SG2TM}(c) = x$, sinon on créé un nouveau topic x dans T tel que nom ne soit pas défini pour x

Chaque sommet concept générique $c \in C_G$ de type $t \leq TA$ est transformé par f_{SG2TM} en une association a de A dont le type est $f^{-1}_{TM2S}(t)$.

Obtention des occurrences et rôles

Les sommets concepts $c' \in C_G$ de type $t_{c'} \leq TPO$ et de marqueur $m_{c'}$ qui sont connectés à c par des relations de type $t_r \leq TLO$ sont transformés par f_{SG2TM} en occurrences de x. On obtient par $occ(x)$ un triplet (tlo, tpo, v) avec $tpo = f^{-1}_{TM2S}(t_{c'})$, $tlo = f^{-1}_{TM2S}(t_r)$ et $v = m$. Ceux de type $t_{c'} \leq C$ qui sont connectés à c par des relations de type $t_r \leq P$ sont transformés par f_{SG2TM} en pointeurs de x vers $f_{SG2TM}(c')$ et on obtient par $occ(x)$ le même triplet (tlo, tpo, v) que dans le cas précédent à ceci près que $tpo = t_{pointeur}$ avec $nom(t_{pointeur}) = Pointeur$. Lorsque $m = *$ et $t_{c'} \leq TPO$ on associe au marqueur générique la valeur d'occurrence $\varnothing$.

Chaque sommet relation $r \in R_G$ de type $t \leq TR$ reliant un sommet concept c de type C et un sommet concept générique c' de type TA est transformé par f_{SG2TM} en une arête e de E (un rôle) dont le type est $f^{-1}_{TM2S}(t)$ reliant $f_{SG2TM}(c)$ à $f_{SG2TM}(c')$.

4.3 Réversibilité

[6] La fonction f_{TM2S} étant injective son inverse est une fonction.

L'application f_{TM2SG} est établie entre l'ensemble TM^A des TM de l'espace d'annotation respectant le niveau modèle et l'ensemble des SG SG^S définis à partir du support S obtenu par l'application de f_{TM2S} sur le niveau modèle. f_{TM2SG} est injective de TM^A vers SG^S : à toute paire de TM différentes elle associe une paire de SG images différents. Par contre, elle n'est pas forcément surjective.

On considère le sous-ensemble $SG^{TM} \subset SG^S$ défini tel qu'un SG de SG^S appartienne à SG^{TM} ssi :

- aucun de ses sommets concepts n'a pour type le plus spécifique un des types fédérateurs : T, C, TA, TPO (ils permettent de distinguer les différentes sortes de type `classe de topic`, `type d'association` ou `type physique d'occurrence` (type de valeur) mais n'ont aucun correspondant au niveau modèle en TM),
- aucun de ses sommets relations n'a comme type le plus spécifique un des types fédérateurs de relations : TLO, TR, P (ils permettent de distinguer les différentes sortes de types `type logique d'occurrence` et `type de rôle` mais n'ont aucun correspondant en TM),
- aucun sommet concept individuel n'a un type inférieur à TA.

f_{TM2SG} établit une surjection (et donc bijection) entre TM^A et SG^{TM}. En effet, si on considère un SG G de SG^{TM},

- Tout sommet concept de marqueur individuel m de type strictement inférieur à C ou TPO a pour antécédent respectif par f_{TM2SG} un sommet topic identifié par m ou une valeur m. En ce qui concerne les concepts individuels de type TA, la question ne se pose pas puisqu'ils n'apparaissent dans aucun SG de SG^{TM}.
- Tout sommet concept générique de type strictement inférieur à C, TA ou TPO a pour antécédent respectif par f_{TM2SG} un sommet topic sans nom, un sommet association ou une valeur $\varnothing$.
- Il faut remarquer que deux sommets de types différents strictement inférieurs à C ou TPO peuvent partager le même marqueur individuel tout en ayant par f_{TM2SG} deux sommets (ou valeurs) antécédent(e)s différent(e)s.
- Toute relation de type strictement inférieur à TR, TLO ou P est binaire, respecte sa signature, et par conséquent a toujours un antécédent par f_{TM2SG} qui peut être une arête représentant un rôle dans le premier cas, ou un type logique d'occurrence dans les deux derniers cas (et plus précisément un pointeur pour le troisième cas).

f_{SG2TM} définit une fonction injective non totale de l'ensemble SG^S vers l'ensemble TM^A qui se révèle être totale de l'ensemble SG^{TM} vers l'ensemble TM^A. La définition de f_{SG2TM} est faite de telle sorte que pour un SG G donné de SG^{TM}, l'antécédent par f_{TM2SG} d'un sommet de G soit l'image par f_{SG2TM} de ce sommet. Par conséquent, si on considère une TM tm $f_{SG2TM} \circ f_{TM2SG}(tm)$ est identique à tm (à l'isomorphisme[7]

[7]On considère que deux TM sont identiques, si elles respectent une relation d'isomorphisme iso_{TM} définie au chapitre `Les Topic Maps`.

près) et $f_{SG2TM} = f_{TM2SG}^{-1}$.

Un SG qui n'a pas été obtenu à partir de f_{TM2SG} mais respecte le support S fourni par f_{TM2S} doit vérifier les conditions énoncées à la définition de l'ensemble SG^{TM} pour être traduit en une TM par f_{SG2TM}. En pratique, pour s'assurer que le résultat des inférences réalisées en SG est traduisible en TM, **il faut que la conclusion de chaque règle appartienne à l'ensemble** SG^{TM}. De ce fait, les graphes produits par applications des règles SG les respecteront aussi et seront traduisibles en TM par f_{SG2TM}.

4.4 Travaux Connexes

La littérature sur les TM présente de manière informelle deux types de démarches pour établir une correspondance entre les TM et le langage RDF. [Moo01] les a l'une et l'autre initialement qualifiées de "modélisation du modèle" et de "transformation du modèle". Puis elles furent respectivement rebaptisées dans [LD01] et [PVG+06] de "correspondance entre objets" et "correspondance sémantique".

Dans la littérature sur les TM, ces deux notions sont définies de façon informelle. On considère qu'un langage du web sémantique permet de représenter des connaissances sous la forme d'*expressions* respectant une grammaire imposée. Ces expressions sont établies au moins à deux niveaux (il peut y en avoir trois) :

- les expressions du niveau *modèle* désignent généralement des *types* ou des *propriétés générales* sur ces types,
- les expressions du niveau *instances* représentent des connaissances assertionnelles (expressions représentant des situations abstraites ou concrètes) qui sont définies en utilisant les types et propriétés déclarés au niveau modèle.

En général, la signification de ces primitives est décrite sous la forme d'une *sémantique intuitive* (en langage naturel) ou d'une *sémantique formelle* (exprimée dans un langage formel tel que la théorie des ensembles ou un langage logique).

Une "correspondance sémantique" entre un langage source et un langage cible établit une bijection entre une partie des expressions du langage source et une partie des expressions du langage cible. En général, une telle correspondance est intéressante lorsqu'il faut préserver la signification des expressions au risque d'obtenir une correspondance incomplète (i.e. certaines expressions des langages source ou cible n'ont pas de correspondant).

Une "correspondance entre objets" définit une fonction injective et totale des expressions du langage source vers les expressions du langage cible (i.e. à chaque expression image du langage cible correspond un seul antécédent dans le langage source). La définition d'une "correspondance entre objets" est intéressante lorsqu'on doit garantir la transformation de chacune des expressions du langage source en autant d'expressions du langage cible sans pour autant que soit préservée la signification des expressions sources.

La définition de ces transformations est principalement motivée par la volonté de

fournir un accès homogène aux connaissances décrites dans les langages source et cible de la transformation.

Dans le cas d'une correspondance entre objets, cet objectif n'est que partiellement atteint. Une telle transformation traduit les expressions du langage source en des expressions qui ne respectent pas forcément les conventions d'écriture du langage cible. Ainsi, la structure des expressions obtenues peut être très différente de celles provenant d'une autre correspondance par objet ciblant le même langage ou naturellement créées par un humain. Si on considère une base de connaissances contenant des expressions du langage cible, l'hétérogénéité de la forme ou de la signification du contenu de la base peut avoir des répercussions sur l'interrogation de la base de connaissances. En effet, la syntaxe d'une requête pour interroger une base de connaissances peut dépendre de la forme des expressions entreposées dans la base. En pareil cas, dès lors que le contenu présente une forme et une signification *hétérogènes*, les irrégularités que présente ce contenu se propagent au niveau de l'interrogation imposant des contraintes sur l'écriture de la requête.

En ce qui concerne les "correspondances sémantiques", les expressions produites sont sémantiquement similaires à leur antécédent dans le langage source et leur syntaxe respecte les conventions d'écriture du langage cible. Ces correspondances étant cependant incomplètes ; certaines connaissances exprimées dans le langage source ne sont pas intégrables dans le langage cible.

Dans les deux cas, les expressions provenant d'une transformation ne sont pas accessibles de la même façon que celles qui respectent les conventions d'écriture du langage cible.

Les transformations présentées dans cette section sont établies du langage des TM vers le langage RDF. Le langage RDF (Resource Description Framework) est un langage de représentation des connaissances permettant de décrire le contenu de ressources web et leurs métadonnées. Un document RDF peut être formulé en XML (respectant une DTD réservée à cet usage) ou dans sa syntaxe abstraite (voir [KC04]). Dans le second cas, on ne parle plus de document RDF mais de "graphe" RDF. Un tel "graphe" est relativement proche de la notion de graphe étiqueté définie en *théorie des graphes*. Dans la suite, les documents RDF sont présentées sous la forme de graphes RDF.

On peut rapidement définir le langage RDF de la façon suivante : soit trois ensembles disjoints U, B et L, un graphe RDF est un ensemble de triplets (s, p, o) avec $s \in U \cup B$, $p \in U$ et $o \in U \cup B \cup L$. L'ensemble U contient les références d'URI RDF, l'ensemble B contient les sommets vierges (ou *blanknodes*) et l'ensemble L des libellés construits sur un alphabet fini. Les éléments s, o et p représentent respectivement le sujet et l'objet entre lesquels le prédicat p établit une relation. La figure 4.19 représente en RDF une acquisition de société par une autre similaire à celle présentée dans la figure 4.18.

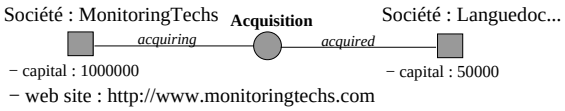


FIG. 4.18 – Une TM.

L.E.A.M.C

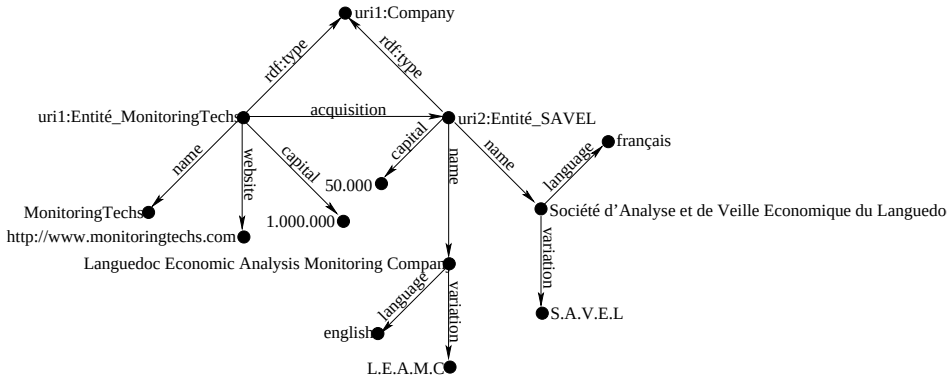


FIG. 4.19 – Un graphe RDF.

Transformation de [Moo01]

Pour établir la correspondance entre objets entre TM et RDF, [Moo01] définit une propriété RDF pour chacun des constituants élémentaires du langage TM : le topic, l'association, la relation d'instanciation, le lien entre un topic et une association, le rôle que joue le topic dans l'association, le topic définissant ce rôle, l'occurrence d'un topic, le nom d'un topic (et sa valeur), le contexte d'existence (scope), les références aux ressources et sources indicatrices d'informations.

L'association binaire entre le topic `MonitoringTech` et le topic `Languedoc...` présentée à la figure 4.18 est définie à partir de cinq topics : deux participent à l'association, deux définissent les rôles et le dernier est le type de l'association. La traduction de cette association selon la correspondance entre objets établie par [Moo01], produit 22 triplets :

trois pour chacun des cinq topics et sept pour l'association elle-même.

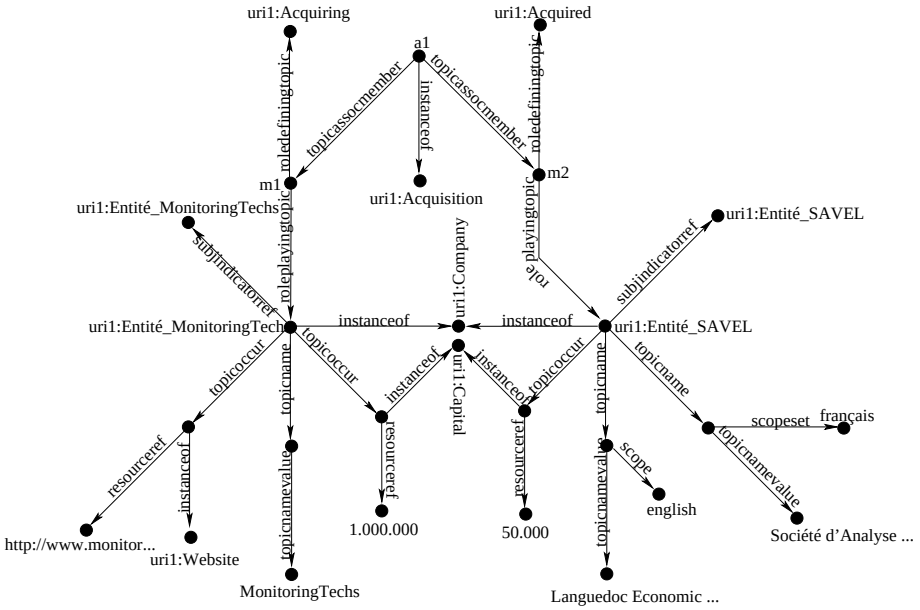


FIG. 4.20 – Le graphe RDF obtenu par la correspondance entre objets établie par [Moo01].

La figure 4.20 présente le graphe RDF correspondant à la TM composée de cette association et des topics qui y participent avec leurs occurrences.

Comme le graphe RDF obtenu à partir de la correspondance entre objets est estimé d'une taille et complexité démesurées par rapport à celui qui aurait été défini pour représenter manuellement les connaissances de la TM, [Moo01] propose une autre transformation des TM vers RDF qui établit une correspondance sémantique partielle.

La démarche consiste à associer une URI du langage RDF à chaque URI du langage TM qui identifie une ressource ou source indicatrice d'information, une ressource RDF à chaque topic et un triplet RDF à une association. La correspondance reste incomplète car les noms, les occurrences et certaines informations nécessaires à la définition de l'association n'ont pas de correspondant en RDF mais bien qu'incomplète, cette correspondance offre une meilleure lisibilité que celle entre objets.

L'initiative entreprise par [Moo01] va inspirer d'autres travaux. Les notions "modélisation du modèle" et "transformation du modèle" seront rebaptisées autrement dans [LD01] qui présente une transformation du premier genre (pour des TM définies en PMTM4) et l'intègre dans un outil de gestion des connaissances fonctionnant en RDF afin d'in-

terroger des connaissances exprimées en TM. [Ogi01] propose une approche similaire à celle de [LD01] en définissant une transformation XSLT[8] de XTM vers RDF. [Gar03] et [GM02] se distinguent des propositions précédentes en proposant une correspondance sémantique partielle des TM (exprimées en TMDM) vers RDF et dont les manques sont complétés de manière spécifique par intervention humaine. Une dernière solution proposée par [Gar05a] consiste à représenter les connaissances exprimées en TM ou RDF dans un langage commun (le langage Q) dans le but de les interroger de façon homogène. Les deux sections suivantes présentent respectivement les correspondances entre objets des TM vers RDF proposées par [LD01] et [Ogi01] et les correspondances sémantiques de [Gar03] et [GM02].

4.4.1 Correspondances entre objets

Transformation de [LD01]

L'objectif des travaux décrits dans [LD01] consiste à interroger des connaissances décrites en TM par un outil de gestion des connaissances fonctionnant en RDF. La transformation est établie à partir des TM PMTM4, le formalisme proposé par [BN01] (voir section 2.3.4), vers le langage RDF.

La démarche consiste à définir des classes (avec rdf :type) et des propriétés RDF pour chaque type de sommets (T A ou S) et d'arcs (AM AS et SC) du formalisme PMTM4, puis à traduire chaque sommet de T, A et S en un sommet de la classe correspondante et chaque arc par la propriété qui lui correspond. Les arcs de AM sont cependant soumis à un traitement particulier, car ils sont étiquetés (ce libellé identifie le rôle que joue le topic de l'une des extrémités de l'arc dans l'association à l'autre extrémité). Ainsi chaque arc de AM est représenté en RDF par un triplet pareillement orienté auquel est attribué par réification l'étiquette identifiant le rôle.

[8] Le langage XSLT permet de définir des transformations entre documents XML (pour plus de détails voir `http://www.w3.org/TR/xslt`).

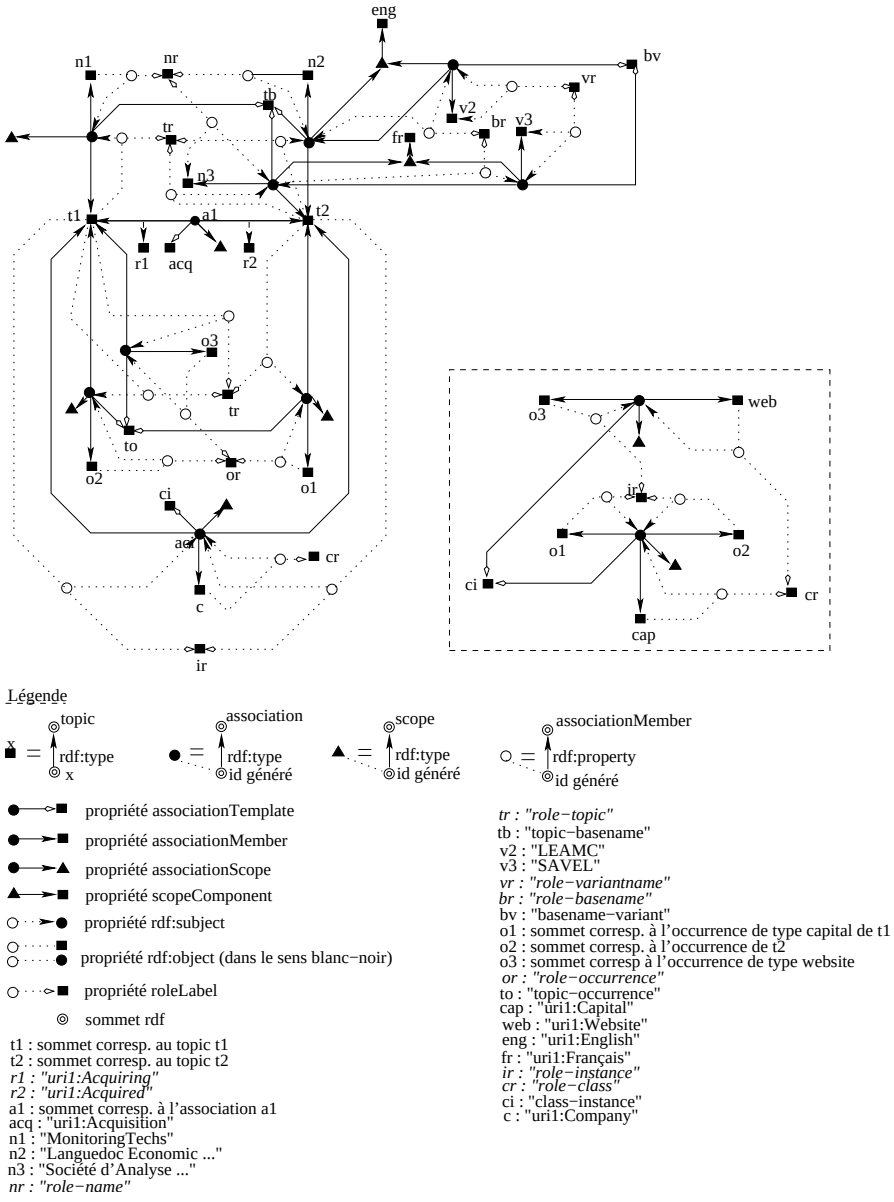


FIG. 4.21 – Le graphe RDF obtenu par la correspondance entre objets établie par [LD01].

La réification en RDF consiste à représenter un triplet RDF (s, p, o) par un sommet RDF x. Trois propriété `rdf :subject`, `rdf :object` et `rdf :property` sont réservées à cet effet. La première est établie entre x et s, la seconde entre x et o et la troisième entre

x et un sommet dont l'identifiant est celui de p.

Dans le cas de [LD01], une propriété particulière (`rôleLabel`) est établie entre x et un sommet portant l'étiquette e de l'arc de AM. Cela signifie que x (qui représente l'arc AM) a pour nom de rôle l'étiquette e.

Le résultat en RDF de cette transformation appliquée sur l'exemple de TM donné à la figure 4.18 se compose de 70 sommets et 219 triplets et est présenté à la figure 4.21.

Il faut toutefois constater que les requêtes (exprimées en F-Logic) pour interroger le résultat en RDF de la traduction d'une TM source doivent respecter une certaine forme qui dépend de la transformation, afin de fournir des réponses identiques à celles qui sont obtenues lorsque la TM source est directement interrogée.

La transformation des TM vers RDF établit entre les deux langages une correspondance entre objets bijective (entre le langage des TM et la portion ciblée de RDF). Cependant, la taille des représentations obtenues en RDF est trop importante ce qui conduit à une lisibilité des informations qui n'est pas assez satisfaisante.

Transformation de [Ogi01]

La transformation d'une TM en RDF présentée dans [Ogi01] définit une correspondance entre objets similaire à celle de [LD01]. Elle est réalisée entre les langages XML associés aux deux formalismes (XTM et RDF) et a été implémentée en XSLT. Elle consiste à définir un vocabulaire de 11 classes et 17 propriétés RDF utilisées pour représenter les constituants de la TM et les rapports qu'ils entretiennent entre eux.

Les classes RDF créées modélisent essentiellement le topic, l'association, l'occurrence et les associations particulières classe-sousclasse et classe-instance. L'attribution d'un domaine de validité (scope) à une association ou occurrence, d'un nom (ou d'une variation) à un topic et d'une occurrence à un topic sont aussi vues comme des associations auxquelles un type particulier est attribué. Les propriétés sont introduites pour représenter les rôles de chacune des associations particulières précédentes. Par exemple, les propriétés `classe` et `instance` sont introduites pour différencier l'instance de sa classe dans la représentation en RDF d'une association classe-instance. La figure 4.22 représente le résultat de la transformation appliquée sur l'exemple de TM donnée à la figure 4.18. La principale différence avec la précédente transformation (cf 4.4.1) réside en ce que cette transformation ne réifie pas le triplet correspondant au lien (association member) entre l'association et le topic pour lui attribuer le nom du rôle. Ce lien est représenté par deux triplets : le premier ayant en sujet le sommet correspondant à l'association, en propriété le nom du rôle et en objet un sommet vierge de type `member` lui même sujet d'un deuxième triplet dont la propriété est `indicatedBy` et l'objet le sommet correspondant au topic.

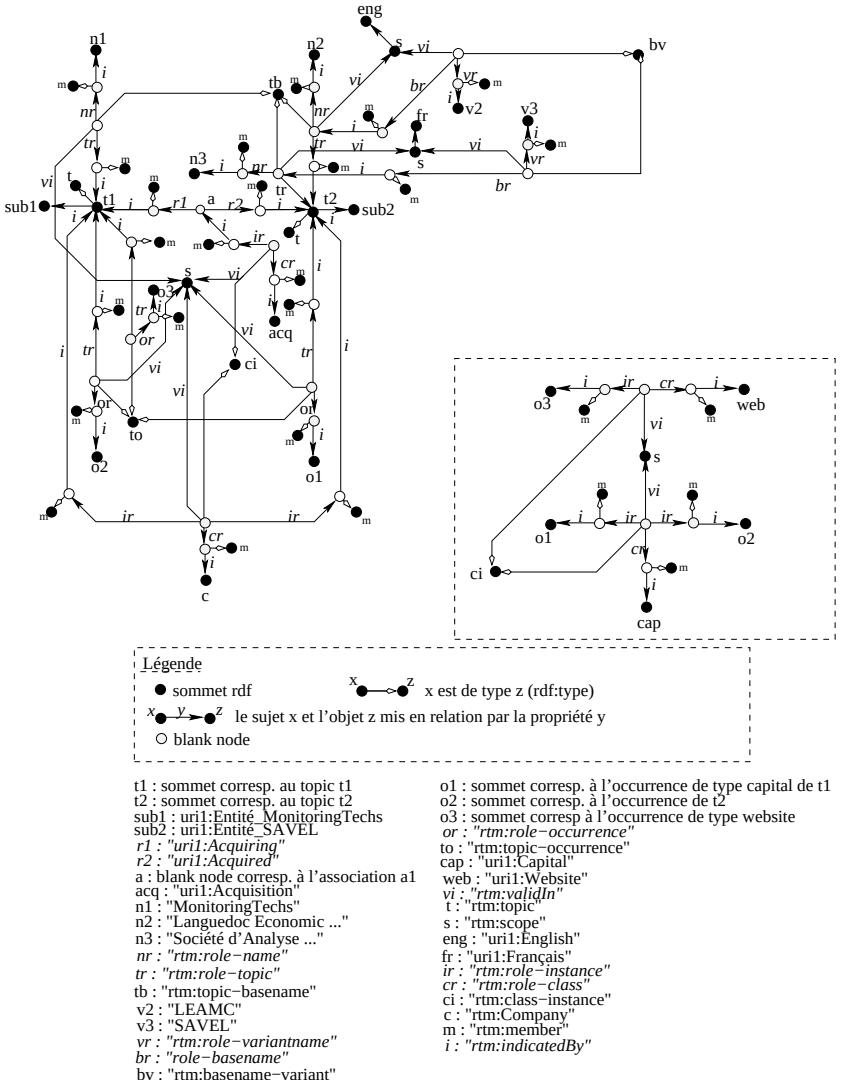


FIG. 4.22 – Le graphe RDF obtenu par la correspondance entre objets établie par [Ogi01].

Chaque élément topic est traduit en un triplet RDF dont la propriété est `rdf :type` et l'objet l'URI de la classe RDF représentant l'élément topic (en l'occurrence `rtm :topic`). Si le sujet du topic est identifié par une URI, elle identifie le sujet du triplet créé pour le topic. Dans le cas contraire, le sujet est un *sommet vierge* (i.e. *blank node*).

Chaque association est traduite en un *sommet vierge*. Un triplet RDF dont la propriété est `rdf :type` permet d'associer à ce *sommet vierge* le type qui représente l'élément

association. Chaque rôle de l'association devient un triplet dont le sujet est le sommet représentant l'association, la propriété est le type du rôle (par exemple, `acquiring`), l'objet est un sommet vierge dont le type correspond à l'élément `member` en XTM. Ce sommet vierge participe en tant que sujet à un second triplet dont l'objet est une ressource identifiée par l'URI du topic jouant le rôle. La propriété de ce triplet dépend de ce qu'identifie l'URI : une source indicative d'information ou le sujet du topic.

Cette transformation a des caractéristiques similaires à celle de [LD01]. Elle établit une correspondance entre objets bijective du langage des TM vers la portion ciblée du langage RDF, mais les représentations en RDF obtenues sont d'une lisibilité insatisfaisante car la quantité de triplets RDF produits est plus importante qu'elle ne l'aurait été si les connaissances avaient été modélisées manuellement.

4.4.2 Correspondances sémantiques

Transformation de [Gar03]

Au lieu de définir une correspondance entre objets, [Gar03] préfère s'en tenir à une correspondance sémantique qui bien qu'incomplète offre l'avantage de conduire à un résultat plus lisible. Les travaux de [Gar03] se limitent à la transformation des associations binaires. Il existe plusieurs manières d'établir le correspondant sémantique en RDF d'un élément de base du langage TM en préservant sa sémantique naturelle. Par exemple, une association binaire (comme l'association de type `Acquisition` de la figure 4.18) pourrait correspondre à deux triplets RDF différents selon que l'on choisit l'un ou l'autre topic participant à l'association comme sujet ou objet du triplet. Dans ce cas, le choix du correspondant RDF dépend de la signification des descriptions faites en TM. La démarche proposée par [Gar03] consiste à assister la transformation d'une analyse humaine qui va fournir les informations nécessaires à l'obtention d'un correspondant RDF unique lorsqu'une ambiguïté apparaît. Ces informations sont exprimées en TM à partir d'un vocabulaire concis (6 PSIs[9]). Par exemple, pour le cas cité plus haut, on précise en utilisant ce vocabulaire lequel des types de rôle (`acquiring` ou `acquired`) que jouent les topics de l'association identifie le topic qui correspond au sujet du triplet RDF résultant (dans notre exemple, le sujet est le topic qui joue le rôle `acquiring`). Si aucun critère de traduction n'est spécifié, la transformation se conforme à un comportement par défaut qui la plupart du temps aboutit à un résultat incomplet (les identifiants de sujets deviennent des URI RDF et le choix sujet/objet est arbitraire pour la traduction des topics d'associations binaires). La figure 4.23 présente le résultat de cette transformation appliqué sur la TM de la figure 4.18.

[9] *PSI* est l'abréviation de *Published Subject Indicator* qui correspond relativement bien à la notion d'URI (cf. Chapitre `Les Topic Maps`).

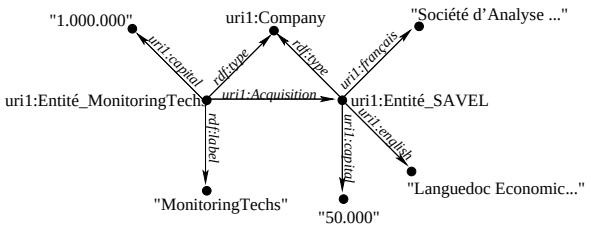


FIG. 4.23 – Le graphe RDF obtenu par la correspondance entre objets établie par [Gar03].

Les transformations du langage TM vers RDF conduisent à un résultat incomplet lorsqu'elles définissent une correspondance sémantique. Lorsqu'elles sont définies entre objets elles produisent des résultats trop volumineux qui, bien qu'intégrables parmi des assertions RDF, ne sont pas pris en compte par le mécanisme d'interrogation de la même manière que le seraient des graphes RDF définis par une main humaine dans le but d'y représenter les mêmes connaissances qu'en TM. Pour résoudre ces problèmes, [Gar05a] propose une autre démarche qui consiste à traduire ces deux langages dans un langage commun, le langage Q introduit en partie 2.3.3, défini tel que le résultat des transformations reste concis et homogène. L'objectif est d'offrir de façon homogène un accès à la connaissance représentée dans chacun des deux langages. En d'autres termes, la syntaxe d'une requête formulée pour interroger les descriptions de ce langage ne doit pas dépendre de leur provenance (elle ne doit pas utiliser de vocabulaire spécifique à la représentation de RDF ou TM). Cet objectif n'est que partiellement atteint et certaines informations (provenant essentiellement des TM) ne sont accessibles qu'en spécifiant un terme qui identifie une notion spécifique au langage (pour interroger certaines associations TM, il peut être nécessaire d'utiliser un terme spécifique réservé à cet usage).

Transformation de [GM02]

Des arguments similaires à ceux fournis par [Gar03] orientent les travaux de [GM02] en faveur d'une correspondance sémantique plutôt qu'entre objets. La démarche, fortement inspirée de celle de [Gar03], consiste à établir une transformation générique qui préserve au mieux en RDF la signification naturelle des éléments TM traduits, tout en laissant la possibilité de la paramétrer de manière spécifique.

La partie générique de la transformation associe à un topic une ressource en RDF dont le nom devient un triplet ayant en sujet le noeud associé au topic, en propriété $rdfs : label$ et en objet le libellé encodant le nom (cette traduction des noms est particulière à cette transformation). La démarche pour représenter en RDF une association est similaire à celle utilisée par [Ogi01] et produit un nombre important de triplets RDF.

De la même façon que dans [Gar03], pour obtenir des résultats plus lisibles, [GM02]

propose un langage qui permet de paramétrer la transformation en prenant en compte la signification naturelle de la TM à traduire.

Comparaison des transformations des TM vers la famille $\mathcal{SG}$

Les trois transformations qui ont été établies pour traduire les trois niveaux *méta*, *modèle* et *instance* de TM dans ITM en un support, un SG, des règles et des contraintes de la famille $\mathcal{SG}$ entretiennent une certaine proximité avec les concepts de "correspondance entre objets" et "correspondance sémantique" définis entre les langages RDF et TM :

- La première transformation a été établie sans prendre en compte les trois niveaux d'ITM ; une TM étant transformée en un SG, qu'elle appartienne au niveau *méta*, *modèle* ou *instance*. Le support quant à lui représente le vocabulaire du langage TM. Cette transformation pourrait être rapprochée du concept de "correspondance entre objet", car elle traduit en un SG l'ensemble des connaissances exprimées en TM sans perdre d'informations. Par contre, elle ne préserve pas forcément la sémantique informelle d'une TM (qui change selon le niveau auquel elle appartient).
- La seconde transformation exploite un peu plus le support afin d'y représenter les connaissances du niveau *méta*. Les TM des niveaux *modèle* et *instances* sont quant à elles toujours représentées dans un SG. Cette transformation a été définie en associant les descriptions du niveau *méta* à des types déclarés dans le support en préservant au mieux leur signification. Cette transformation n'est cependant pas tout à fait complète ; car certaines définitions au niveau *méta* n'ont pas d'image dans le support. Son caractère partiel et le fait qu'elle accorde un peu plus d'importance à la sémantique la rapproche d'une "correspondance sémantique". Cependant, cette transformation peut aussi être considérée comme une "correspondance entre objets" ; puisqu'elle traduit en un SG l'intégralité des niveaux *modèle* et *instance* sans accorder d'importance à leur sémantique. Elle se situe donc à mi-chemin entre la correspondance "sémantique" et celle "entre objets".
- La dernière transformation consiste à traduire différemment chacun des trois niveaux de TM. Les niveaux *méta*, *modèle* et *instance* sont respectivement envoyés sur les primitives intrinsèques à la famille $\mathcal{SG}$, le support et le SG. Certaines propriétés des types déclarées au niveau *modèle* sont traduites en règles et contraintes SG. Cette transformation peut accuser certaines "pertes" mais elle reste, parmi les trois, celle qui préserve le mieux la sémantique des TM. Elle se rapproche donc beaucoup plus nettement que la seconde du concept de correspondance "sémantique".

Cette dernière transformation offre le meilleur compromis entre la conservation de la sémantique des représentations à transformer, la lisibilité/simplicité des représentations obtenues et l'étendue de son domaine (la portion du langage sur laquelle elle s'applique). Cette transformation est utilisée par le *service d'enrichissement et de validation* ; la

solution développée pour *opérationaliser* les connaissances représentées dans ITM en s'appuyant sur les capacités de contrôle et de raisonnement de la famille $\mathcal{SG}$. Ce service est présenté au chapitre suivant.

Chapitre 5

Les services de raisonnements

Le chapitre précédent présentait les transformations définies entre le langage des TM et la famille $\mathcal{SG}$; dont une est dédiée aux TM de niveau modèle et l'autre aux TM de niveau instance. Ce chapitre décrit le système qui a été élaboré à partir des transformations précédentes pour doter ITM des capacités d'inférence et de contrôle de la famille $\mathcal{SG}$. Il a été baptisé "service de *validation* et d'*enrichissement*" car sa fonction principale consiste à *valider* et *enrichir* les annotations ajoutées à ITM. Le terme *validation* signifie que le contenu de l'annotation est contrôlé et celui d'*enrichissement* indique que ce contenu est complété de nouvelles connaissances déduites à partir de celles initialement présentes dans l'annotation.

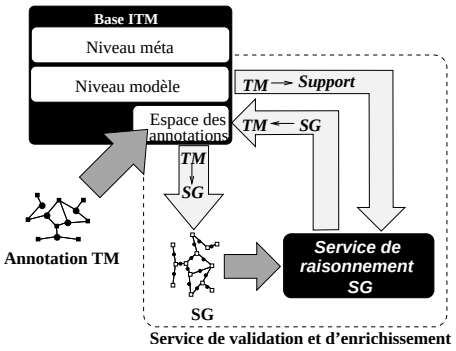


FIG. 5.1 – L'architecture du service de validation et d'enrichissement.

La figure 5.1 présente l'architecture générale du système. Le service de *validation* et d'*enrichissement* (*SVE*) y est représenté en pointillé. Le service de raisonnement $\mathcal{SG}$ (représenté en noir) dispose de sa propre base d'annotations dont le contenu est une copie, enrichie par application de règles d'inférence, d'une portion "cohérente" de l'espace des

annotations d'ITM. Le choix de la portion "cohérente" est déterminé par l'ordre d'ajout des annotations SG à la base ; toute nouvelle annotation rendant la base incohérente est rejetée. Ceci permet d'assurer la monotonie des raisonnements effectués sur cette base.

A l'initialisation, le *SVE* utilise les transformations présentées au chapitre précédent pour traduire le niveau modèle d'ITM en un support et des ensembles de règles et de contraintes chargées dans le service de raisonnement $\mathcal{SG}$. Lorsqu'une annotation est ajoutée à ITM, le *SVE* la traduit en un SG puis l'envoie au service de raisonnement $\mathcal{SG}$ qui suit un processus itératif en quatre étapes de *contrôle*, d'*intégration*, d'*enrichissement* et de *validation* avant d'accepter ou de rejeter l'annotation. Le service de raisonnement $\mathcal{SG}$ gère aussi l'ajout de règles et de contraintes ; ou le retrait de contraintes s'il est nécessaire de modifier les ensembles de règles et de contraintes après initialisation.

La première section de ce chapitre définit le service de raisonnement $\mathcal{SG}$. La seconde section décrit la partie de ITM qui a été utilisée pour concevoir le *SVE*. La troisième section présente le *SVE* et la dernière section est consacrée à deux projets au cours desquels le *SVE* a été utilisé : le projet PressIndex dédié à la veille médiatique et le projet Eiffel concernant la valorisation de ressources touristiques.

5.1 Service de raisonnement $\mathcal{SG}$

Le service de raisonnement $\mathcal{SG}$ est une base notée $B_{\mathcal{SG}}$ de connaissances $\mathcal{SG}$ offrant un service d'*interrogation*, un service d'*ajout contrôlé* d'annotation, de règle ou de contrainte et un service de retrait de contrainte. La sous-section suivante énonce les propriétés de *cohérence* et de *structure* que doit toujours vérifier $B_{\mathcal{SG}}$ pour rester fiable. La deuxième sous-section décrit le fonctionnement des trois services $B_{\mathcal{SG}}$; et la troisième sous-section définit en détail les mécanismes utilisés par ces services et notamment les quatre étapes de *contrôle*, d'*intégration*, d'*enrichissement* et de *validation* du service d'ajout contrôlé.

5.1.1 La base

La base $B_{\mathcal{SG}} = (S, \mathcal{R}, \mathcal{C}^+, \mathcal{C}^-, G, i)$ de connaissances $\mathcal{SG}$ est constituée d'un support S à partir duquel sont définis un graphe G dit *la base d'assertions*, un ensemble $\mathcal{R}$ de règles et deux ensembles $\mathcal{C}^-$ et $\mathcal{C}^+$ de contraintes négatives et positives. Le support S, les règles $\mathcal{R}$ et les contraintes $\mathcal{C}^+$ et $\mathcal{C}^-$ sont fournies à l'initialisation de la base de connaissances. i est une fonction qui associe à tout sommet de G un identifiant : i est définie de l'ensemble des sommets de G dans un ensemble $\mathcal{L}_G$ contenant d'une part les marqueurs individuels de I et d'autre part des libellés différents de ceux de I (et construits sur un alphabet fini). Deux sommets s et s' de G désignent une même entité *ssi* $i(s) = i(s')$.

Le graphe G est un SG qui respecte deux ensembles de propriétés. D'une part les propriétés dites de *cohérence* :

1. les types qu'il utilise sont définis dans le support S ;

2. les relations respectent leur signature ;
3. il ne viole aucune contrainte négative $\mathcal{C}^-$;

d'autre part des propriétés dites de *structure* :

4. i est totale et injective pour l'ensemble des sommets de G ;
5. il est sous forme normale ;
6. il est sous forme irredondante ;
7. il est saturé par rapport à l'ensemble de toutes les règles $\mathcal{R}$.

5.1.2 Les services

Trois services sont fournis à partir de la base $B = (S, \mathcal{R}, \mathcal{C}^+, \mathcal{C}^-, G, i)$: un service d'interrogation, un service de retrait de contraintes et un service d'*ajout contrôlé*. Chaque graphe G_R retourné par ces services est accompagné d'une fonction notée $i_{[G_R]}$ résultant de la restriction de i aux sommets de G_R.

Le service d'interrogation renvoie les connaissances assertionnelles de la base qui répondent à une requête exprimée sous la forme d'un graphe SG (i.e les sous-graphes de G qui sont les images par projection de la requête).

Le service de retrait de contraintes supprime de $\mathcal{C}^-$ ou $\mathcal{C}^+$ une contrainte C. Dans le cas où C est une contrainte positive, il renvoie les sous-graphes de G qui violent C afin d'indiquer les portions de la base qui violaient C et sont "réparées" par la suppression de C.

Le service d'ajout contrôlé réalise quatre opérations différentes selon la nature de l'argument fourni (que l'on note X). Ce dernier peut être une règle, une contrainte négative, une contrainte positive ou une annotation SG. Chacune de ces opérations consiste à ajouter X respectivement aux ensembles $\mathcal{R}$, $\mathcal{C}^-$ et $\mathcal{C}^+$ ou au graphe G en modifiant éventuellement la base pour qu'elle respecte les propriétés de *structure*. On s'assure ensuite que la base obtenue vérifie les propriétés de *cohérence* :

- Dans le cas où elle les vérifie, le graphe X est *accepté* et on retourne éventuellement le sous-graphe de la base qu'il a fallu ajouter ou modifier pour que la base respecte les propriétés de *structure* après l'ajout de X. Le sous-graphe retourné peut être accompagné d'une structure contenant les contraintes positives de $\mathcal{C}^+$ violées ou "réparées" par l'ajout de X ; le viol d'une contrainte positive ne remettant pas en cause la cohérence de la base.
- Dans le cas où la base ne les vérifie pas, elle est restaurée (i.e. elle retrouve l'état précédant l'ajout contrôlé) et l'ajout contrôlé retourne le graphe X accompagné des raisons de son rejet ; c'est à dire les violations de contraintes négatives.

Les trois sections suivantes décrivent respectivement les services d'interrogation de la base, de retrait d'une contrainte et d'ajout contrôlé ; ce dernier service utilisant des procédures subalternes qui sont détaillées dans la quatrième section 5.1.3.

Interrogation

Étant donnés une base de connaissances $B = (S, \mathcal{R}, \mathcal{C}^+, \mathcal{C}^-, G, i)$ et un SG Q construit sur le même vocabulaire, l'interrogation de la base consiste à retourner tous les sous-graphes G_Q images d'une projection de Q dans G, chacun d'entre eux étant accompagné d'une fonction $i_{[G_Q]}$ correspondant à la restriction de i aux sommets de G_Q.

Retrait de contraintes

Étant donnés une base de connaissances $B = (S, \mathcal{R}, \mathcal{C}^+, \mathcal{C}^-, G, i)$ et une contrainte C de $\mathcal{C}^-$ ou $\mathcal{C}^+$, le *retrait* de C consiste à supprimer la contrainte C de $\mathcal{C}^-$ ou $\mathcal{C}^+$.

Dans le cas où C est une contrainte positive, on construit l'ensemble R_C composé des couples (G_π, π) où G_π est le sous-graphe de G image de chaque projection π de la condition de C qui ne peut être étendue à C. Les graphes G_π de cet ensemble indiquent les portions de la base d'assertions qui violaient C lorsqu'elle appartenait à $\mathcal{C}^+$. Ces sous-graphes de G sont dits "réparés" par le retrait de C.

Le couple $(\emptyset, V)$ avec $V = \{(C, \emptyset, R_C)\}$ est retourné par le service.

Ajout contrôlé d'un graphe X (règle, contrainte ou annotation)

On considère une base de connaissances $B = (S, \mathcal{R}, \mathcal{C}^+, \mathcal{C}^-, G, i)$, le SG X et une fonction i_X de l'ensemble des sommets de X dans $\mathcal{L}_G$. La fonction i_X n'est définie que lorsque X est une annotation et associe un identifiant à chaque sommet de X. Deux sommets différents s et s' de X désignent alors des entités identiques s'ils partagent un même marqueur individuel, s'ils sont reliés par un lien de coréférence ou si $i_X(s) = i_X(s')$; sinon ils désignent des entités différentes.

L'ajout contrôlé consiste à enchaîner 4 procédures, chacune prenant en entrée le résultat de la procédure précédemment réalisée. Ces quatre procédures sont présentées en détail dans la partie 5.1.3 suivante.

(1) le contrôle de X : On vérifie si les signatures et les types des sommets concepts et relations de X sont dans le support S. Le cas échéant X est mis sous forme normale (quelle que soit la nature de X).

(2) l'intégration : Si X est une annotation, elle est ajoutée et fusionnée à G. Sinon X étant une règle, une contrainte négative ou une contrainte positive, elle est respectivement ajoutée à l'ensemble idoine $\mathcal{R}$, $\mathcal{C}^-$ ou $\mathcal{C}^+$.

(3) l'enrichissement par $\mathcal{R}$: Si X est une annotation la procédure d'enrichissement est réalisée à partir du sous-graphe de la base d'assertions induit par les sommets crées à l'étape précédente. Dans le cas où X est une règle, X a été appliquée sur G selon toutes les projections possibles de son hypothèse et la procédure d'enrichissement est réalisée sur l'ensemble des sommets ajoutés dans la base lors de ces applications.

La procédure d'enrichissement permet de calculer la fermeture de G à partir des règles de $\mathcal{R}$.

(4) la validation : Si X est une règle ou une annotation, cette étape consiste à détecter les sous-graphes de la base qui violent les contraintes positives de $\mathcal{C}^+$. Par contre, si X est une contrainte positive, on ne cherche qu'à détecter les sous-graphes de la base qui violent X. Dans le premier cas, la procédure de validation est réalisée à partir du sous-graphe de la base induit par les sommets créés dans la base aux étapes précédentes ; dans le second, elle est appliquée à toute la base d'assertions.

La figure 5.2 illustre le déroulement de l'ajout contrôlé. Le déclenchement des quatre étapes - (1) de contrôle, (2) d'intégration, (3) d'enrichissement et (4) de validation - dépend de la nature de X. Dans le cas où X est une contrainte positive les étapes 1, 2 et 4 sont réalisées. Lorsque X est une contrainte négative seules les étapes 1 et 2 sont déclenchées.

La procédure de vérification des contraintes négatives $\mathcal{C}^-$ est réalisée à l'issue de chaque étape 1, 2 ou 3 ; cette vérification n'ayant lieu après l'étape 2, que si X est une annotation ou une contrainte négative. On s'assure ainsi que l'annotation SG résultant de l'étape 1 et la base d'assertions issue des étapes 2 ou 3 ne violent aucune contrainte négative ; et lorsque X est une contrainte négative on vérifie que la base d'assertions ne viole pas X. Cette procédure est détaillée dans la partie `Vérification des contraintes négatives` de cette même section. Le déclenchement des étapes 2, 3, et 4 est conditionné au résultat des étapes précédentes : l'étape suivante n'étant réalisée que si l'étape courante n'a pas conduit à la violation d'une contrainte négative.

A la fin des étapes 2 et 3 du mécanisme d'ajout contrôlé, la base d'assertions est préservée sous forme normale et i reste totale et injective, ce qui assure à la base d'être sous forme irredondante. Après l'étape 3, si aucune contrainte négative n'a été violée, la modification de la base est définitive, et le résultat de l'étape 4 est retourné. Il contient les sommets inférés durant l'opération et les éventuelles violations ou réparations de contraintes positives. Dans le cas où une violation de contrainte négative a lieu, on revient au graphe G.

La validation de la base d'assertions permet de signaler un manque de connaissances (violation de contrainte positive) ou la réparation d'un tel manque (satisfaction d'une contrainte positive violée avant l'ajout) mais ne remet pas en cause la "cohérence" de la nouvelle base. Les contraintes positives sont vues comme des déclencheurs d'avertissement de connaissances incomplètes et non comme des "incohérences". Cela laisse la possibilité de "compléter" par ajout contrôlé les connaissances manquantes. L'étape de validation est réalisée en fin de processus car l'étape d'enrichissement a pu "réparer" une absence initiale de connaissances qui aurait conduit au viol d'une contrainte positive.

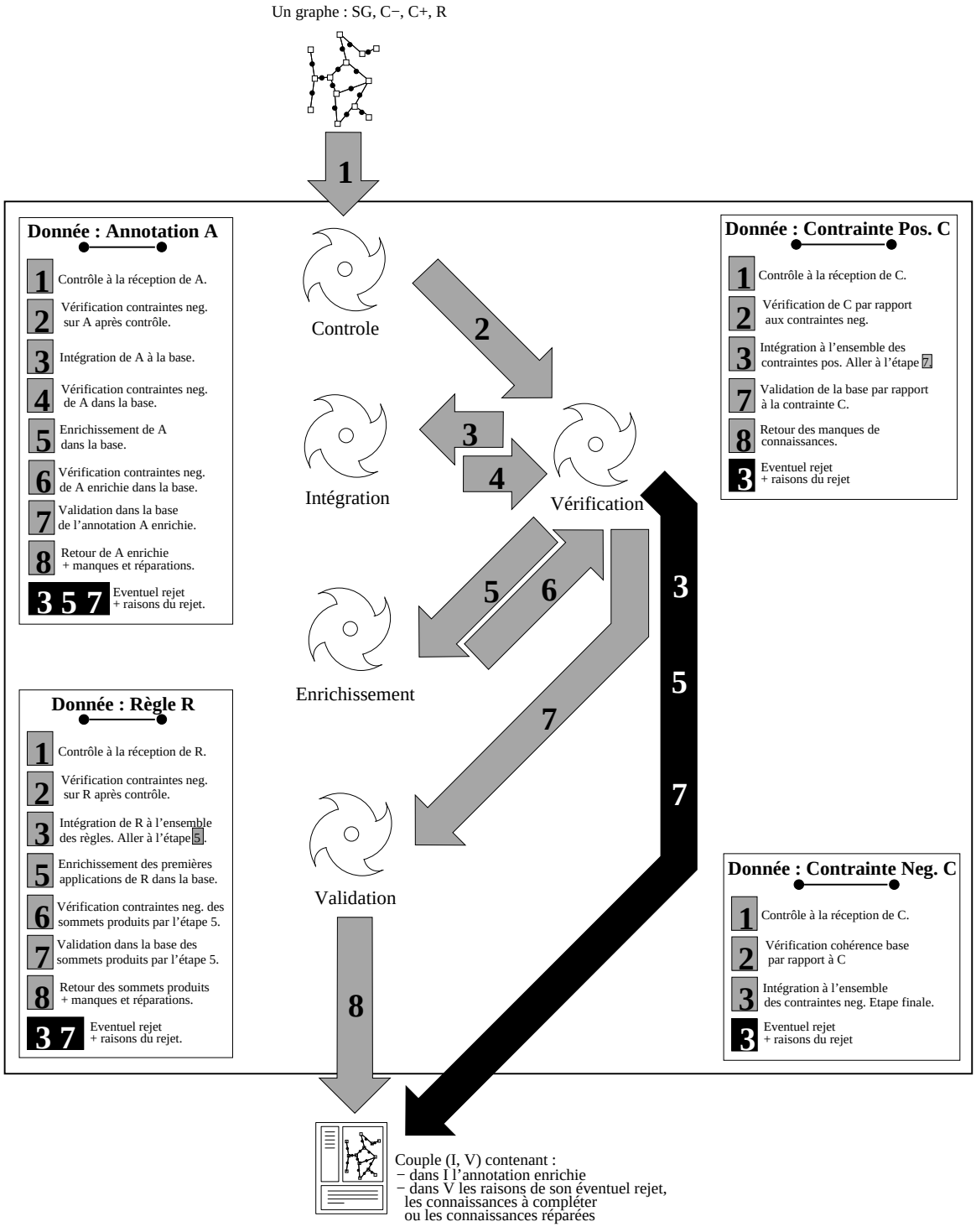


FIG. 5.2 – Étapes et déroulement de l'ajout contrôlé.

Une contrainte négative violée par une annotation ou règle X (à l'étape 1), est assurée de l'être par le graphe G après intégration (étape 2) ou enrichissement (étape 3). En effet, tout SG résultant d'éventuelles fusions de sommets après une union disjointe est toujours plus spécifique que les SG à partir desquels il a été obtenu. De ce fait, toute projection dans l'un de ces derniers existe également dans le premier. Or, les étapes 2 et 3 d'intégration et d'enrichissement spécialisent le graphe G par fusions ou ajouts de sommets. Si X est une annotation, l'étape 2 réalise l'union disjointe puis la fusion de X à G et dans le cas où X est une règle ou une annotation, l'étape 3 réalise une série d'applications de règles sur G. On en conclut qu'une contrainte négative violée après l'étape i est assurée de l'être après l'étape $i+1$ (avec $1 \leq i < 3$). La détection en amont d'une telle violation conduit au rejet de X et permet d'éviter de déclencher inutilement l'étape 3 qui peut être coûteuse en temps.

Lorsque X est une contrainte positive et qu'une contrainte négative C s'y projette[1] à l'étape 1, si un sous-graphe K de G satisfait X et qu'il existe une projection dans K de la partie condition[2] de X alors K viole aussi C. Tout ajout contrôlé de règle ou d'annotation pour satisfaire X conduit donc à une violation de C. La contrainte positive ajoutée étant insatisfiable, elle devient inutile et est rejetée.

5.1.3 Procédures subalternes

Cette section décrit la maintenance de G sous forme irredondante en préservant l'injectivité de i et détaille chaque procédure de *contrôle*, d'*intégration*, d'*enrichissement*, de *vérification* et de *validation* réalisée par l'ajout contrôlé.

Conservation de l'injectivité de i

i est une fonction injective et totale de l'ensemble des sommets de G dans l'ensemble $\mathcal{L}_G$ des libellés telle que (1) l'image par i d'un sommet individuel soit son marqueur individuel et (2) celle attribuée à un sommet générique est un libellé qui n'est pas un marqueur individuel. L'injectivité de i signifie que deux sommets différents de G désignent toujours des entités différentes. Cette caractéristique des sommets de G pourrait être représentée en $\mathcal{SG}$ en reliant chaque sommet concept de G aux autres par un lien $\neq$ (évoqué au troisième chapitre `La famille` $\mathcal{SG}$). Cependant, pour mieux illustrer l'importance de la fonction i, on considère un type de relation dif, une contrainte $\mathcal{C}^{\circ}_{dif}$ interdisant à dif d'être réflexive (cf. figure 5.3) et on attribue à dif la signification suivante : deux sommets entre lesquels dif apparaît désignent des entités différentes. Une relation de type

[1] X est un SG bicoloré. La projection de C sur X est réalisée comme la projection conventionnelle sans tenir compte de la coloration.

[2] Une contrainte positive est toujours satisfaite par un SG si sa partie condition ne s'y projette pas. On souhaite ici ne pas tenir compte de ce cas.

dif ne modifie pas le comportement du mécanisme standard de projection (dif étant un type déclaré dans le support), alors qu'un lien $\neq$ est interprété d'une façon particulière au cours de la projection : c'est pour cette raison que le type de relation dif est considéré ici.

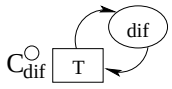


FIG. 5.3 – Contrainte négative astreignant dif à rester irreflexive.

Par définition, i impose à tous les sommets de G de désigner des entités différentes. Cela équivaut à ce qu'après l'ajout, chaque sommet concept soit relié par la relation dif à chacun des autres dans G. Cette propriété garantit que G est toujours sous forme *irredondante*. En effet, si chaque sommet est relié par une relation dif à tous les autres, il ne peut être projeté sur un de ses voisins puisque pour l'être dif devrait boucler autour de ce dernier (ce qu'interdit $\mathcal{C}^{\circ}_{dif}$). Il n'existe donc aucune projection du graphe G dans un de ses sous-graphes stricts : G est donc sous forme irredondante.

Bien que le type de relation dif et la contrainte $\mathcal{C}^{\circ}_{dif}$ soient définissables en $\mathcal{SG}$, pour des raisons d'efficacité, elles ne sont pas intégrées au support et à l'ensemble $\mathcal{C}^{-}$ de contraintes. En effet, une telle relation dif[3] apparaîtrait un nombre $n(n-1)$ de fois dans la base d'assertion, n étant le nombre de sommets concepts de G. Ce nombre n'étant pas négligeable, il aurait une mauvaise incidence sur les performances des opérations. Il est donc préférable de contraindre la forme de G, des règles de $\mathcal{R}$ et des contraintes positives de $\mathcal{C}^{+}$ afin de toujours assurer la satisfaction de cette contrainte.

Pour ce faire, il est nécessaire d'empêcher la fusion des sommets qui désignent des entités différentes. De telles fusions signifieraient que deux sommets désignent des entités à la fois différentes et identiques (puisque la relation dif bouclerait autour du sommet résultant de la fusion) ; ce qui violerait la contrainte $\mathcal{C}^{\circ}_{dif}$ et rendrait la base incohérente. Pour garantir la satisfaction de $\mathcal{C}^{\circ}_{dif}$, le lien de coréférence ne doit pas apparaître :

(1) entre deux sommets différents de G ayant des images différentes par i ou

(2) entre deux sommets frontières différents de la conclusion d'une règle de $\mathcal{R}$ ou de la partie obligation d'une contrainte positive de $\mathcal{C}^{+}$.

Ainsi, la mise sous forme normale, l'application d'une règle ou la réparation d'une violation de contrainte positive dans G ne conduisent pas à la fusion de deux sommets différents.

[3]Cette remarque s'applique aussi au lien $\neq$.

Contrôle d'un graphe X

La procédure de contrôle prend en entrée un SG, une règle ou une contrainte X accompagné de la fonction i_X. Si X est un graphe bicoloré représentant une contrainte ou une règle, i_X n'est définie pour aucun des sommets de X. Dans tous les cas, i_X est complétée de telle sorte que pour tout sommet individuel s de marqueur m, $i_X(s) = m$.

La figure 5.4 illustre un exemple d'annotation SG et de SG bicoloré.

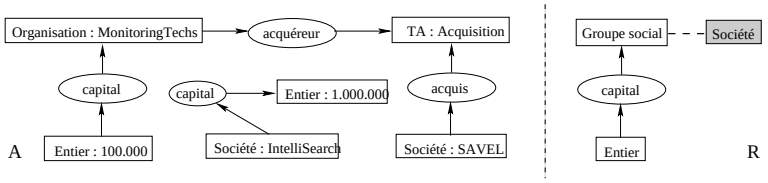


FIG. 5.4 – Une annotation SG ou une règle SG en entrée de la procédure de contrôle.

On commence par vérifier si X respecte quatre conditions syntaxiques. Si une de ces conditions n'est pas respectée, un couple (C, V_C) est ajouté à l'ensemble V, C étant soit un symbole soit une contrainte[4] qui identifie la condition insatisfaite. Ensuite, le mécanisme s'arrête et retourne le couple (X, V).

Ces conditions sont les suivantes :

(1) Le type de chaque sommet du graphe X est dans le support : si cette condition n'est pas vérifiée, C_τ est le symbole indiquant que cette condition n'est pas vérifiée. V_{C_τ} est un ensemble contenant uniquement le couple $(\emptyset, G_\tau)$ où G_τ est un SG qui rassemble les sommets invalides de X. Le premier élément du couple désigne habituellement une projection qui n'est pas nécessaire dans ce cas ci.

(2) Les types des sommets qui désignent des entités identiques ne sont pas incomparables : si cette propriété est invalide, $C_\tau^=$ signifie que cette condition n'est pas vérifiée et l'ensemble $V_\tau^=$ contient les couples $(\emptyset, G_\tau^=)$ où le graphe $G_\tau^=$ rassemble les sommets de X de types incomparables reliés par un lien de coréférence. La figure 5.5 présente le SG $G_\tau^=$ qui viole cette contrainte. Comme `Groupe social` correspond à une classe de topic et `Acquisition` à un type d'association, ils sont tous deux incomparables.

(3) Aucun sommet s n'est relié à un sommet s' par un lien de coréférence si $s \neq s'$, $i_X(s) = i(s)$ et $i_X(s') = i(s')$. Ces sommets ont une image par i et désignent donc dans G des entités différentes. Dans le cas où cette condition n'est pas satisfaite,

[4]Il faut noter que si C est une contrainte, elle n'appartient pas pour autant à l'un des ensembles $\mathcal{C}^-$ et $\mathcal{C}^+$. En effet, elle prend la forme d'une contrainte de la famille $\mathcal{SG}$ pour préserver l'homogénéité du résultat retourné par l'ajout contrôlé. Ainsi toute violation peut être expliquée par une contrainte où au pire un symbole dont la signification est connue.

chaque paire de sommets non valides est regroupée dans le graphe $G^{=}_{\neq}$ du couple $(\emptyset, G^{=}_{\neq})$ ajouté à l'ensemble $V^{=}_{\neq}$. $C^{=}_{\neq}$ désigne la contrainte négative $\mathcal{SG}$ interdisant à deux sommets différents d'être reliés par un lien de coréférence. La figure 5.5 représente la contrainte $C^{=}_{\neq}$ et un SG $G^{=}_{\neq}$ qui la viole.

(4) Les relations du graphe X respectent les signatures définies dans S. Si tel n'est pas le cas, le sous-graphe $G^{\neq}_{\sigma}$ induit par chaque relation invalide de type r et ses sommets concepts voisins forment un couple $(\emptyset, G^{\neq}_{\sigma})$ qui est ajouté à l'ensemble V^{r}_{σ}. C^{r}_{σ} désigne la contrainte positive $\mathcal{SG}$ qui impose à toute relation de type r de n'être reliée à un concept que s'il est d'un type permis par la signature $\sigma(r)$. Une contrainte de ce type pour la relation `acquéreur` est représentée par le SG C_{σ} à la figure 5.5.

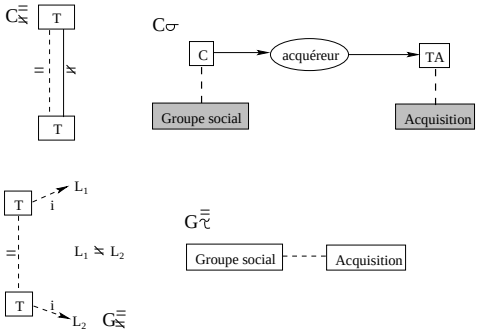


FIG. 5.5 – Représentation des deux contraintes $C^{=}_{\neq}$ et C_{σ} et de deux violations $G^{=}_{\tau}$ et $G^{=}_{\neq}$

Rejet ou normalisation

Si une étape a conduit à la création de couples $(C_{\tau}, V_{C_{\tau}})$, $(C^{=}_{\tau}, V^{=}_{\tau})$ $(V^{=}_{\neq}, C^{=}_{\neq})$ et $(C^{r}_{\sigma}, V^{r}_{\sigma})$, ils sont ajoutés à V et la procédure de contrôle s'arrête en retournant (X, V). Sinon, lorsque l'étape de contrôle conduit à un succès, le graphe X est normalisé. Le résultat de la normalisation dépend de la nature de X. Dans tous les cas, la normalisation produit un graphe X_F obtenu en *fusionnant* dans X les sommets concepts qui désignent des entités identiques. Lorsque X est bicoloré (contrainte ou règle), la procédure de fusion conduit à la définition d'un second graphe X'_F dans lequel ces deux sommets ne sont fusionnés que s'ils sont de même couleur. La figure 5.6 présente les graphes obtenus.

Pour normaliser un graphe X, on considère tous les couples de sommets s_0 et s_1 de X de types respectifs t_0 et t_1 et de marqueurs m_0 et m_1. On réalise la fusion de s_0 et s_1 lorsqu'ils sont individuels et que $m_0 = m_1$; ou bien génériques et reliés par un lien

de coréférence. Dans le cas où X est bicoloré, la génération du graphe X'_F impose aux deux sommets d'être de même couleur. Le sommet s résultant de la fusion est de type t_0 si $t_0 \leq t_1$ (et t_1 si $t_1 < t_0$) et de marqueur m_0. i_{X_F} (ou $i_{X_{F'}}$) assigne alors à s le libellé attribué à s_0 par i_X lorsqu'il existe. Ce procédé est relancé sur le graphe résultant de la fusion jusqu'à stabilisation.

La procédure retourne : le graphe X_F éventuellement accompagné de X'_F.

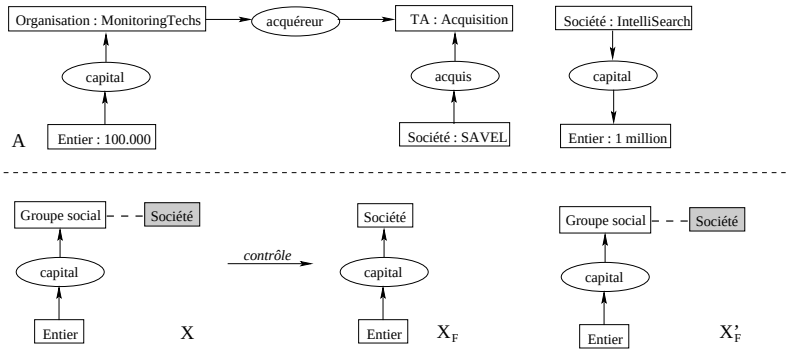


FIG. 5.6 – Le résultat de la procédure de contrôle : L'annotation n'a pas changé. Dans le cas où le SG est bicoloré, deux graphes X_F et X'_F sont produits.

Insertion d'une annotation X dans la base G

Soit X l'annotation fournie par la procédure de contrôle et G la base d'assertions dont un exemple est donné à la figure 5.7.

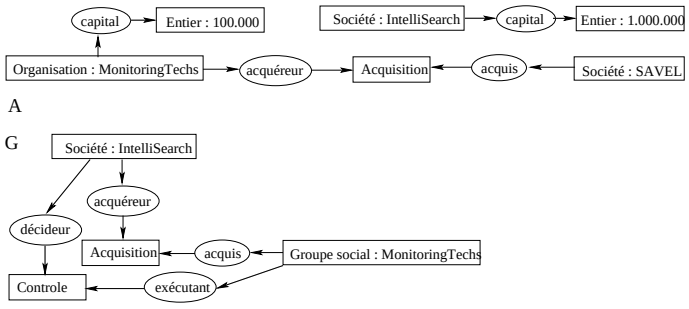


FIG. 5.7 – L'état de l'annotation et de la base d'assertions avant insertion.

X est ajouté au graphe G en fusionnant les sommets concepts qui désignent des entités identiques. Après fusion, i attribue à tout nouveau sommet générique de G un

libellé de $\mathcal{L}_G$ qui n'a pas d'antécédent dans G par i ; ceci permet de maintenir i totale et injective et par voie de conséquence préserve la base d'assertions sous forme irredondante. Cette étape retourne le sous-graphe de G induit par les sommets ajoutés ou fusionnés.

La figure 5.8 présente l'état de la base d'assertions résultant de l'insertion de l'annotation. Le sous-graphe coloré rassemble les sommets ajoutés ou fusionnés. Il fait partie du sous-graphe entouré de pointillés qui est retourné par la procédure.

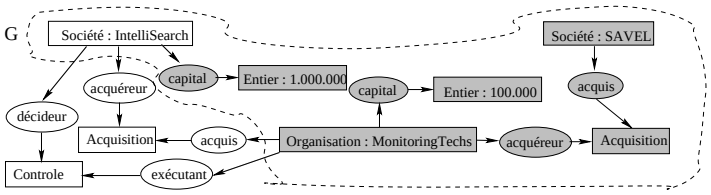


FIG. 5.8 – L'état de la base d'assertions après insertion.

Enrichissement de G à partir d'un sous-graphe A

Soit A le sous-graphe de G induit par les sommets ajoutés au cours de l'ajout contrôlé avant le déclenchement de la procédure d'enrichissement. Un exemple d'un tel sous-graphe est donné en grisé à la figure 5.8. L'enrichissement de G à partir de A consiste à appliquer dans G les règles dont l'hypothèse se projette sur au moins un sommet provenant soit de A, soit d'une des applications réalisées par l'enrichissement en cours ; et ceci tant que ces dernières ajoutent de nouvelles connaissances.

La figure 5.9 présente trois exemples de règles utilisées pour enrichir la base d'assertions de la figure 5.8. La règle R_1 permet de déduire qu'un groupe social qui en acquiert un autre, contrôle ce dernier et la règle R_2 signifie que le contrôle s'exerce de façon transitive. La règle R_3 est une règle de spécialisation de type qui permet de déduire qu'un groupe social détenant un capital est une société.

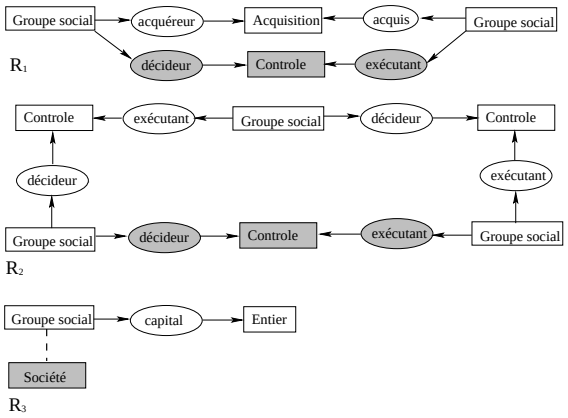


FIG. 5.9 – Un exemple de trois règles.

Pour toute règle R, on s'intéresse aux projections π de l'hypothèse de R dans la base qui projettent au moins un sommet de l'hypothèse sur la portion ajoutée (le sous-graphe A). Pour chacune de ces projections, on vérifie ensuite que l'application de la règle selon cette projection apporte effectivement de l'information (on s'assure qu'il n'existe pas de projection de la conclusion de R dans la base qui projette chaque sommet frontière sur son image par π). Si tel est le cas, on applique[5] la règle selon π. On réitère ce processus, en considérant que la portion ajoutée est maintenant celle formée par les nouveaux sommets inférés et ceci jusqu'à ce que cette portion soit vide (ce qui se produit forcément au bout d'un temps fini si les règles sont à saturation finie - voir chapitre sur la famille $\mathcal{SG}$).

On désigne par I le sous-graphe de la base induit par les sommets inférés au cours de cette étape et par $A \cup I$ le sous-graphe de la base composé de I et de A. La fonction i attribue à chaque sommet $s \in I$ un libellé de $\mathcal{L}_G$ qui n'a aucun autre antécédent par i.

La figure 5.10 présente la base d'assertions résultant de l'enrichissement. Le sous-graphe I est représenté en gris et $A \cup I$ rassemble les sommets gris et noirs. Le sommet de marqueur `MonitoringTechs` a été spécialisé par application de la règle R_3.

[5] on raccorde la conclusion aux images par π des sommets frontières

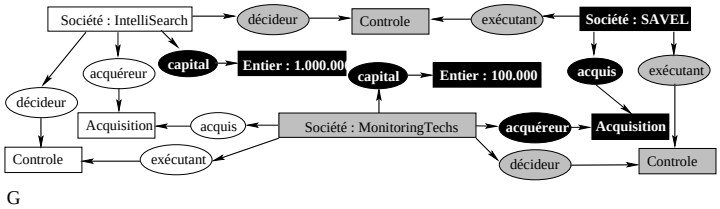


FIG. 5.10 – La base d'assertion modifiée par l'étape d'enrichissement.

A la fin de cette étape, l'enrichissement retourne le SG $A \cup I$.

Vérification des contraintes négatives

La vérification des contraintes négatives dépend de la nature de l'argument X fourni à l'ajout contrôlé. Le couple (K, V) est retourné par la vérification à l'issue de chacune des situations suivantes.

(1) Lorsque l'argument X de l'ajout contrôlé est une contrainte négative, on considère chaque projection π de X sur G et $K = \emptyset$.

(2) Lorsque l'argument X est une contrainte positive, une règle ou une annotation SG, l'étape 1 retourne un SG X_F éventuellement accompagné de X'_F. Dans ce cas, pour chaque contrainte négative $C \in \mathcal{C}^-$ on considère chaque projection π de C sur X_F. Si X'_F est présent on a $K = X'_F$, sinon on a $K = X_F$.

(3) Dans le cas où l'ajout contrôlé ne concerne qu'une annotation SG provenant de l'étape 2 ou 3, on dispose du sous-graphe A de G induit par les sommets ajoutés à ces étapes. Dans ce cas, $K = A$ et on considère les projections π de C sur au moins un sommet de A et au moins un sommet de G.

Pour chacune des situations précédentes, on ajoute dans V un couple (C, V_C) tel que V_C soit l'ensemble des couples (π, G_π) formés par chacune des projections π précédentes et de son graphe image G_π.

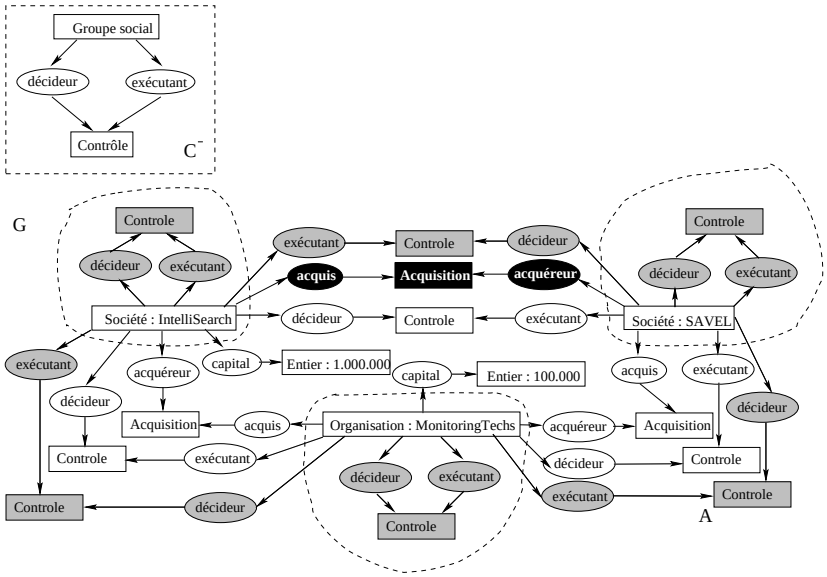


FIG. 5.11 – L'identification après l'étape d'enrichissement des violations de la contrainte négative C

La figure 5.11 illustre la vérification de la contrainte négative C signifiant que "un groupe social ne peut être à la fois décideur et exécutant dans une même association de contrôle". Cette vérification a lieu après les étapes d'intégration et d'enrichissement appliquées sur l'annotation (représentée en noir) et sur la base de la figure 5.10. Chaque sous-graphe entouré de pointillés correspond à l'image d'une projection π de C dans la base et forme avec π un couple de l'ensemble V_C de (C, V_C).

Validation

Le déroulement de la procédure de validation dépend de la nature de l'argument de l'ajout contrôlé. L'objectif de l'étape de validation est de déterminer les sous-graphes de la base qui "réparent" ou violent des contraintes positives à la fin de l'ajout contrôlé. Dans la suite, le symbole π_V désigne une projection qui identifie un sous-graphe de la base à l'origine d'une *violation* de contrainte positive ; tandis que le symbole π_R en désigne une qui identifie un sous-graphe de la base à l'origine d'une *réparation* de contrainte positive.

(1) Lorsque l'argument est une contrainte positive C, la procédure vérifie la validité de G par rapport au SG C'_F accompagnant le SG C_F résultant de l'étape 1. Dans ce cas, on considère les projections π_V sur G de la condition de la contrainte C'_F qui ne peuvent être étendues à une projection de C'_F entière dans G.

(2) Lorsque l'argument est une annotation SG, on dispose du sous-graphe A de la base d'assertions induit par les sommets ajoutés à G au cours des étapes précédentes. Dans ce cas, on considère pour chaque contrainte positive $C^+ \in \mathcal{C}^+$ et sa partie *condition* C

(a) les projections π_V de C dans la base d'assertions qui incluent au moins un sommet de A et ne peuvent être étendues à une projection de C^+ entière et

(b) la restriction π_R aux sommets de C des projections dans la base d'assertions de la contrainte C^+ entière ; chacune de ces projections devant inclure au moins un sommet de A qui n'est l'image par projection d'aucun sommet de C.

Pour chacune des situations précédentes, on ajoute dans V un triplet (C, V_C, R_C) pour chaque contrainte C concernée tel que chacun des deux ensembles V_C et R_C contienne respectivement les couples (π_V, H_V) et (π_R, H_R) formés par chacune des projections π_V et π_R précédentes ainsi que leurs graphes images H_V et H_R. L'image H_V est le sous-graphe de G qui viole la contrainte C. H_R est le sous-graphe de la base d'assertions qui viole C quand le sous-graphe A est absent de la base. Or A étant le sous-graphe induit par les sommets ajoutés durant l'ajout contrôlé, il "répare" le sous-graphe H_R à l'origine de la violation. A la fin de l'étape de validation, le couple (A, V) est retourné.

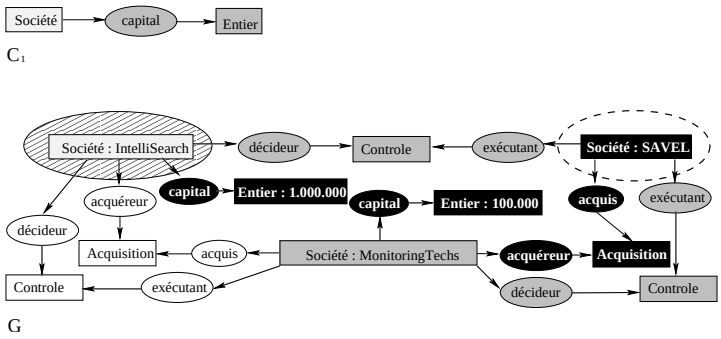


FIG. 5.12 – Une annotation qui à la fois (1) viole la contrainte positive C_1 et (2) répare un sous-graphe de la base violant cette contrainte.

La figure 5.12 illustre la validation de la base d'assertions de la figure 5.10. Le sous-graphe hachuré violait la contrainte positive C_1 avant ajout et ne la viole plus a présent. Il est ajouté avec la projection dont il est l'image à l'ensemble R_{C_1} de (C_1, V_{C_1}, R_{C_1}). Par contre, le sous-graphe entouré de pointillés viole C_1 et est ajouté dans V_{C_1} avec la projection qui l'identifie.

Caractère incrémental des opérations

Les opérations de mise sous forme normale, de saturation par rapport à $\mathcal{R}$ et de vérification des contraintes sont réalisées de manière incrémentale au cours de l'ajout contrôlé d'une règle ou d'une annotation.

Avant l'ajout contrôlé, le graphe G est sous forme normale, saturé par rapport à $\mathcal{R}$, valide par rapport aux contraintes négatives et à l'origine d'un nombre n de violations de contraintes positives. Ces opérations sont dites incrémentales car elles concentrent leur traitement sur la portion qui conduit la base à ne plus respecter une de ces propriétés. Or, comme le montre la suite de cette section, cette portion ne peut être que le sous-graphe induit par les sommets ajoutés au cours de l'ajout contrôlé. Dans la suite, le graphe G désigne le graphe de la base d'assertions avant l'ajout contrôlé, A le sous-graphe éventuellement ajouté à G à l'étape 2, I le sous-graphe induit par les sommets inférés à l'étape 3 et $A \cup I$ celui formé par les sommets de A et de I. Les graphes G_2 et G_3 désignent respectivement le graphe représentant la base d'assertions à la fin des étapes 2 et 3 ; G_3 étant plus spécifique que G_2 lui même plus spécifique que G.

Si G_2 n'est plus sous forme normale, comme A est sous forme normale, cela ne peut être dû qu'à au moins un sommet s de A et s' de G désignant une même entité. On peut donc se contenter de fusionner dans G_2 les sommets de A qui forment de telles redondances avec ceux de G.

Si le graphe G_2 n'est plus saturé, cela signifie que des applications de règles qui ajoutent de nouvelles connaissances peuvent être réalisées. Or, G étant saturé la projection de l'hypothèse H_0 d'une règle R_0 dans G ne peut déclencher son application puisque la conclusion C_0 de R_0 est déjà connectée à l'image de cette hypothèse (C_0 se projette dans G de telle sorte que l'image de chaque sommet frontière selon cette projection soit identique à celle qui lui est attribuée par la projection de H_0). Une application de règle n'est donc susceptible d'être déclenchée que lorsque la projection de H_0 est réalisée sur au moins un sommet de la base n'appartenant pas à G, c'est à dire un sommet parmi ceux de A ou parmi ceux de I produits par les applications de règles qui viennent d'être effectuées. Pour saturer G_2 en G_3, on concentre la recherche d'une projection de l'hypothèse d'une règle sur le sous-graphe induit soit par les sommets de A soit par les sommets de I issus des applications précédentes, de telle sorte que chaque projection d'hypothèse ait lieu sur au moins un de ces sommets.

Si le graphe G_2 ou G_3 ne respecte plus les contraintes négatives, cela signifie qu'au moins une contrainte négative C se projette dans G_2 ou G_3. Or, C ne peut se projeter uniquement sur G dans G_2 ou G_3 puisque G est valide. C se projette donc au moins sur un sommet de A dans G_2 ou de $A \cup I$ dans G_3. On concentre donc la vérification des contraintes négatives de $\mathcal{C}^-$ en cherchant une projection de la contrainte sur au moins un sommet de ces sous-graphes.

En ce qui concerne les contraintes positives, pour chaque contrainte positive C de

$\mathcal{C}^+$, il existe avant ajout n projections π_i (i de 1 à n) de la condition H de C dans G qui ne peuvent être étendues à C entière. S'il existe une projection π de la condition H de C dans G_3 :

(1) soit π a lieu uniquement dans G :

(a) dans ce cas si cette projection ne peut être étendue dans G à C entière elle fait partie des projections π_i et la violation a déjà été signalée.

(b) Dans le cas où π peut être étendue à une projection π' de C entière uniquement dans G, elle n'identifie pas de violation puisque C est vérifiée.

(c) Dans le cas où π peut être étendue à une projection π' de C entière sur au moins un sommet de $A \cup I$, il existe une projection π_k parmi les projections π_i identique à π puisque π n'est pas extensible à C uniquement dans G (au moins un sommet de $A \cup I$ est nécessaire). Or comme π est extensible à C entière, il en est de même pour π_k qui n'identifie donc plus de violation dans G_3 (la violation est réparée par l'ajout). On en conclut que la violation d'une contrainte positive C est réparée par un ajout, si la partie condition de C ne se projette dans G_3 sur aucun sommet de $A \cup I$; et la partie obligation sur au moins l'un d'entre eux.

(2) soit π concerne au moins un sommet de $A \cup I$ et dans ce cas, si π ne peut être étendue à C entière elle identifie une nouvelle violation due au présent ajout contrôlé. On en conclut que pour déterminer les violations dues à l'ajout, il suffit de chercher les projections π sur au moins un sommet de $A \cup I$ qui ne peuvent être étendue à C entière.

5.2 L'outil ITM

Cette section présente l'état existant de l'outil ITM, les services qu'il met à disposition et ceux qui ont été élaborés sans s'appuyer sur le service de raisonnement $\mathcal{SG}$.

Les connaissances d'ITM sont exprimées sous la forme de TM qui sont stockées dans un SGBD à partir duquel ITM offre différents services d'acquisition et d'exploitation de ces connaissances.

L'acquisition des connaissances est réalisée par l'intermédiaire d'interfaces à base de formulaires "dynamiquement contrôlés" par l'ontologie au niveau modèle, par un service d'importation de connaissances au format XTM qui permet d'intégrer des connaissances issues d'autres outils[6], et enfin par un module d'annotation semi-automatique qui permet d'alimenter ITM avec la connaissance extraite de documents non structurés et semi-structurés.

Ce dernier service est conçu à partir de méthodes du traitement du langage naturel [Ama06]. Il utilise une "cartouche linguistique" qui établit les motifs syntaxiques à repérer dans les textes. Cette cartouche a été conçue par une société spécialisée qui a mené

[6]En particulier, cela permet d'intégrer des ontologies de domaine construites à l'aide d'éditeurs spécialisés.

une expertise linguistique du domaine sur lequel porte l'acquisition de connaissances. Les motifs syntaxiques de la cartouche sont définis conformément à un vocabulaire structuré qu'il est nécessaire de mettre en correspondance avec celui d'ITM pour pouvoir représenter dans la base les informations extraites [ALM05]. Pour chaque document analysé, le service d'annotation produit une TM. Par exemple, la TM de la figure 5.15 qui présente une association `Acquisition` pourrait être extraite d'un document présentant les expressions "MonitoringTechs a acquis SAVEL", "La société IntelliSearch dispose d'un capital d'un million d'euros" et "Le capital de MonitoringTechs est de 100.000 euros".

L'exploitation des connaissances peut se faire par navigation graphique en parcourant le réseau Topic Maps. Une méthode de recherche par chaîne de caractères permet d'explorer les noms, les occurrences textuelles mais aussi les documents textes (externes à ITM) référencés par des topics. Une recherche multi-critères est également proposée. Elle permet, de retrouver des topics en spécifiant à l'aide de formulaires leur type, les types d'occurrences possédées et les types d'association parmi lesquelles ils jouent un rôle.

Il faut noter que cet ensemble de services constitue une API qui peut être utilisée comme un service web.

5.2.1 Espaces de travail

La gestion par l'outil ITM des espaces de travail est présentée au chapitre `Les Topic Maps`. La TM associée à l'espace dédié aux annotations est désignée par $tm_{\mathcal{A}}$ et la TM du niveau modèle commun à tous les espaces de travail est tm_{modele}. Les figures 5.13 et 5.14 présentent respectivement une partie du contenu de l'espace dédié aux annotations et du niveau modèle.

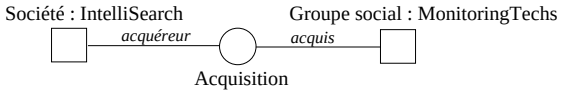


FIG. 5.13 – Exemple de contenu de l'espace des annotations.

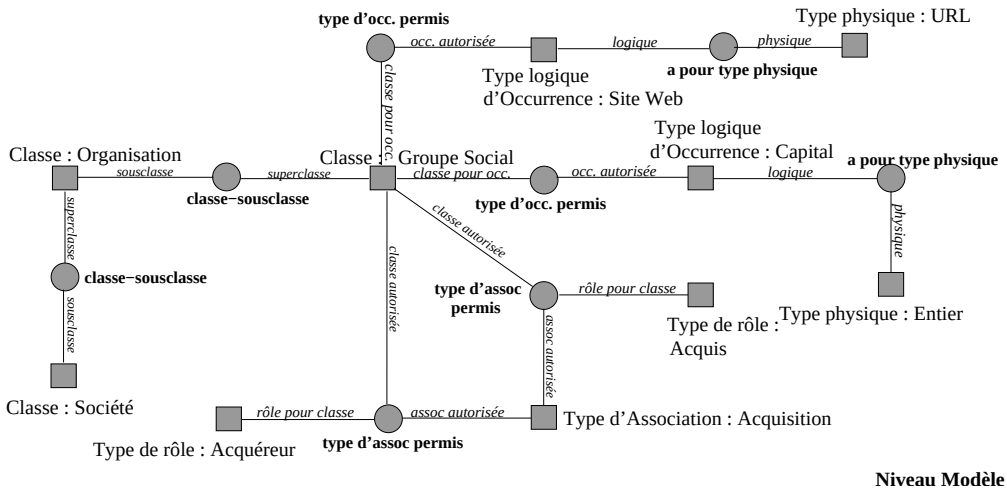


FIG. 5.14 – Une partie du niveau modèle.

5.2.2 Opérations

Deux opérations α et σ d'*ajout spécialisant* et de *suppression généralisante* de TM permettent de faire évoluer la base de connaissances d'ITM. Ces opérations ont été définies au cours de ce travail de thèse à partir des opérations élémentaires de création ou suppression d'un élément de TM dont dispose ITM.

Ajout spécialisant

L'ajout spécialisant $\alpha(tm_S, tm_C)$ d'une TM *source* tm_S dans une TM *cible* tm_C consiste à créer dans tm_C les éléments de tm_S absents de tm_C puis à éventuellement spécialiser ceux déjà présents dans tm_C. Les topics étant identifiés par leurs noms[7] , l'ajout spécialisant fusionne les topics de la TM source avec ceux de même nom dans la TM cible en attribuant au topic résultant de la fusion le type du topic le plus spécifique parmi ceux fusionnés.

[7]Rappel : dans ITM, la fonction *nom* associe un PSI (sorte d'identifiant unique) aux topics d'une TM.

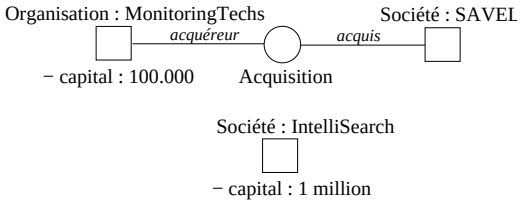


FIG. 5.15 – Une TM.

La figure 5.16 illustre le résultat de l'ajout spécialisant $\alpha(tm, tm_{\mathcal{A}})$ dans l'espace des annotations présenté à la figure 5.13 de la TM tm de la figure 5.15 représentant l'acquisition de la société `SAVEL` par la société `MonitoringTechs`. La classe `Groupe social` du topic `MonitoringTechs` dans la base est spécialisée en la classe `Organisation` du topic de même nom dans la TM, et l'occurrence de type logique `capital` a été ajoutée au topic de nom `IntelliSearch`.

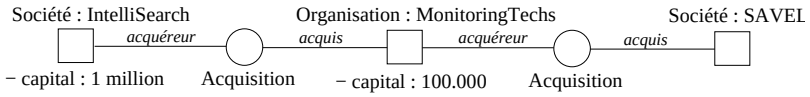


FIG. 5.16 – Résultat de l'ajout spécialisant d'une TM dans l'espace des annotations.

Suppression généralisante

La suppression généralisante $\sigma(tm_S, tm_C)$ d'une TM *source* tm_S dans une TM *cible* tm_C consiste à supprimer ou généraliser les éléments de tm_C qui se trouvent dans tm_S. Un topic ou une association x n'est supprimé dans tm_C que si son type, ses occurrences, ses rôles et son nom sont identiques à ceux de son homologue dans tm_S, sinon il est conservé dans la TM résultante et éventuellement généralisé.

La figure 5.17 illustre le résultat de la suppression généralisante d'une TM A_S dans l'espace des annotations présenté à la figure 5.16. A_S représente l'acquisition de la `Société SAVEL` par le `Groupe social MonitoringTechs`. La classe `Société` du topic `MonitoringTechs` dans la base est généralisée en la classe `Groupe social` du topic portant le même nom dans A_S, tandis que les autres topics, associations et occurrences sont supprimés.

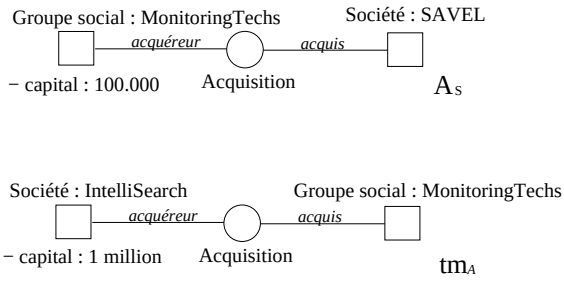


FIG. 5.17 – Résultat de la suppression généralisante d'une TM A_S dans l'espace des annotations.

Propriétés

L'ajout spécialisant et la suppression généralisante entretiennent des relations particulières avec la relation de spécialisation $\leq_{\mathcal{SG}}$ du formalisme $\mathcal{SG}$; comme illustré à la figure 5.18.

Soit deux TM complètes tm_0 et tm_1 et les SG G_0 et G_1 résultant de leur traduction par f_{TM2SG}, le résultat de l'ajout spécialisant de tm_0 dans tm_1 est tel que le SG résultant de sa traduction par f_{TM2SG} est plus spécialisé que G_0 et G_1.

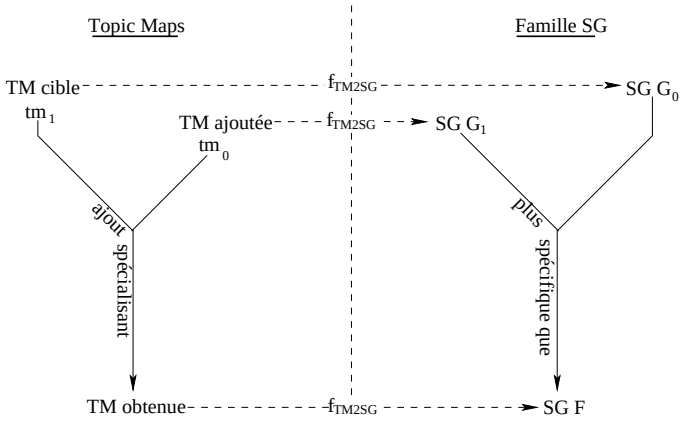


FIG. 5.18 – Propriétés de spécialisation $\mathcal{SG}$ de l'ajout spécialisant.

La suppression généralisante vérifie la propriété suivante. Le résultat tm de la suppression généralisante d'une TM tm_0 dans une TM tm_1 est tel que G_1 est plus spécialisé que le SG résultant de la traduction par f_{TM2SG} de tm.

Interrogation

ITM fournit deux opérations élémentaires d'interrogation l'une notée i_t retourne un ensemble de topics et l'autre notée i_a un ensemble de tm composées d'une unique association, de ses roles et de ses topics voisins. Chaque topic ou association peut porter des occurrences. On fournit aux opérations i_t et i_a la description des topics et associations recherchés en leur passant respectivement en argument un *profil* de topic et d'association.

Le profil d'un topic permet de décrire le type, le nom, les occurrences portées par le topic recherché, le profil des associations auxquelles il participe et son rôle dans ces associations. Le profil d'une association décrit le type et les occurrences de l'association recherchée ainsi que le profil des topics participant à cette association. Le profil d'un topic ou d'une association x intervenant dans la définition d'un autre profil est limité à la description du type et des occurrences de x (et de son nom, si x est un topic). La recherche ne porte donc que sur les associations complétées de leur topic, sur les topics participant à au moins une association, et sur les topics isolés.

Ces opérations élémentaires sur la base sont accessibles à partir du web (en services web) et permettent aussi de définir des services plus évolués d'interrogation mis à disposition de l'utilisateur à l'aide d'interfaces graphiques. Parmi ces services on distingue essentiellement le service de recherche par nom, le service de recherche multi-critères et le service de navigation.

En ce qui concerne les deux premiers, le résultat d'une requête est une liste de topics ou d'associations qui correspondent à la recherche. Le service de recherche par nom prend en argument une chaîne n de caractères et retourne les topics dont le nom contient n ou portant une occurrence dont la valeur contient[8] n. Le service de recherche multi-critères donne la possibilité à l'utilisateur de définir précisément le profil des topics et associations recherchées que les fonctions i_t et i_a prennent en argument.

Le service de navigation est le service par défaut qu'ITM offre à ses utilisateurs. Il permet de consulter le voisinage d'un topic. A chaque étape ce service affiche le topic sur lequel l'utilisateur se trouve, son nom, ses occurrences et les associations dans lesquelles il intervient. L'utilisateur peut ensuite se déplacer vers un nouveau topic situé au bout d'une de ces associations. Une méthode de recherche généralement admise consiste à utiliser les services de recherche par nom ou multi-critères pour centrer l'affichage sur un topic autre que les voisins du topic courant ; puis continuer la recherche dans le voisinage du topic choisi en utilisant le service de navigation.

5.2.3 Notion d'annotation

[8] Il existe plusieurs méthodes de recherche du libellé n : on peut considérer les noms ou les valeurs d'occurrences qui sont identiques à n, qui commencent par n , se terminent par n, ou contiennent n.

L'espace des annotations $E_{\mathcal{A}}$ dispose d'une métadonnée particulière utilisée pour identifier les *annotations* de $tm_{\mathcal{A}}$. Dans la suite, cette métadonnée est représentée par la fonction *annotations* qui associe à chaque élément de $tm_{\mathcal{A}}$ un sous-ensemble de $L_{\mathcal{A}}$ où $L_{\mathcal{A}}$ est un ensemble de libellés.

Définition d'une annotation

tm est une annotation de $E_{\mathcal{A}}$ ssi tm est la plus grande TM incluse dans $tm_{\mathcal{A}}$ telle qu'il existe un libellé $\mathbb{A} \in L_{\mathcal{A}}$ pour lequel tout $x \in elt(tm)$ vérifie $\mathbb{A} \in annotations(x)$. Intuitivement, lorsque la fonction *annotations* est correctement définie, elle associe à x l'ensemble des annotations auquel x appartient.

Étant donné un libellé $\mathbb{A} \in L_{\mathcal{A}}$, la fonction tm associe au libellé $\mathbb{A}$ la TM $(T, A, E, occ, nom, type)$ construite à partir de $tm_{\mathcal{A}}$. $tm(\mathbb{A})$ est telle que pour tout $x \in elt(tm_{\mathcal{A}})$ si $\mathbb{A} \in annotations(x)$ alors $x \in elt(tm(\mathbb{A}))$. Pour tout libellé d'annotation $\mathbb{A}$, $tm(\mathbb{A})$ est assurée d'être une TM ssi la fonction *annotations* respecte les quatre conditions suivantes :

- si $o \in occ(x)$ alors $annotations((o, x)) \subseteq annotations(x)$ avec $x \in T \cup A$,
- si $n = nom(t)$ alors $annotations((n, t)) \subseteq annotations(t)$ avec $t \in T$,
- si $e = (t, a) \in E$ alors $annotations(e) \subseteq annotations(t) \cap annotations(a)$,
- si $\tau = type(x)$ alors $annotations((\tau, x)) \subseteq annotations(x)$ avec $x \in T \cup A \cup E$.

Topic et association *périphériques*

Un topic ou une association x d'une TM tm_0 est en *périphérie* d'une TM tm_1 ssi x appartient à tm_1 et au moins un élément parmi le type de x, le nom de x (si x est un topic), les occurrences de x et les rôles de x apparaissant dans tm_0 n'apparaît pas dans tm_1. x est dit *périphérique* dans tm_0 et tm_1 ssi dans tm_0 x est en périphérie de tm_1 et dans tm_1 x est en périphérie de tm_0.

Annotations *adjacentes*

Deux annotations $\mathbb{A}$ et $\mathbb{A}'$ sont dites *adjacentes* ssi l'intersection de $elt(tm(\mathbb{A}))$ avec $elt(tm(\mathbb{A}'))$ n'est pas vide et ne contient que des topics ou des associations qui sont *périphériques* dans $tm(\mathbb{A})$ et $tm(\mathbb{A}')$.

Annotations *disjointes*

Deux annotations $\mathbb{A}$ et $\mathbb{A}'$ sont disjointes ssi les ensembles $elt(tm(\mathbb{A}))$ et $elt(tm(\mathbb{A}'))$ sont disjoints.

Opérations sur les annotations

A partir des opérations d'ajout et de suppression de TM on peut définir l'*ajout et la suppression d'une annotation*. Une troisième opération appelée *disjonction* permet d'identifier une TM de $tm_{\mathcal{A}}$ en tant que nouvelle annotation.

L'ajout d'une TM tm en tant qu'annotation de $E_{\mathcal{A}}$ identifiée par $\mathbb{A}$ consiste à ajouter tm dans $tm_{\mathcal{A}}$ tel que $\mathbb{A}$ appartienne à l'ensemble associé par *annotations* à chaque élément ajouté ou spécialisé.

La suppression de l'annotation identifiée par $\mathbb{A}$ consiste à réaliser une suppression généralisante de $tm(\mathbb{A})$. La fonction *annotations* est modifiée par l'opération : les éléments de $tm(\mathbb{A})$ supprimés ne sont plus dans le domaine de *annotations* et les ensembles associées par *annotations* aux éléments conservés[9] ne contiennent plus le libellé $\mathbb{A}$.

La disjonction $\delta_{\mathcal{A}}$ d'une TM tm incluse dans $tm_{\mathcal{A}}$ consiste à attribuer à tm un nouveau libellé d'annotation $\mathbb{A}$ tel que $\mathbb{A}$ soit *disjointe* ou *adjacente* à toute autre annotation $\mathbb{A}' \in L_{\mathcal{A}}$. Lorsque la disjonction produit une nouvelle annotation $\mathbb{A}$ à partir d'une TM tm formée des éléments de diverses annotations, la fonction *annotations* est modifiée telle que chacune de ces annotations devienne *adjacente* à $\mathbb{A}$. La disjonction $\delta_{\mathcal{A}}(tm, \mathbb{A})$ est obtenue :

- en ajoutant $\mathbb{A}$ à $annotations(x)$ où x est un topic ou une association de $tm_{\mathcal{A}}$ en *périphérie* de tm (et *annotations* est définie pour x) ;
- en attribuant l'ensemble $\{\mathbb{A}\}$ comme image par *annotations* de tous les autres éléments de tm.

5.3 Service de validation et d'enrichissement d'ITM

Le service *SVE* de validation et d'enrichissement d'ITM s'appuie sur le service de raisonnement $\mathcal{SG}$ pour contrôler et compléter le contenu des annotations. Il est composé de l'espace réservé aux annotations $E_{\mathcal{A}}$, de tm_{modele} et du service de raisonnement $B_{\mathcal{SG}} = (S, \mathcal{R}, \mathcal{C}^+, \mathcal{C}^-, G, i)$.

Un statut de validité *invalide*, *incomplet* ou *complet* est attribué à chaque annotation $\mathbb{A}$ de $E_{\mathcal{A}}$; sachant qu'une annotation de statut *incomplet* ou *complet* est dite *valide*.

La figure 5.19 illustre le fonctionnement du *SVE*. A l'initialisation le contenu de tm_{modele} est traduit en un support, un ensemble de règles et un ensemble de contraintes chargés dans $B_{\mathcal{SG}}$. Toute annotation $\mathbb{A}$ de $E_{\mathcal{A}}$ est traduite en un SG A, lui même ajouté à $B_{\mathcal{SG}}$ par le service d'*ajout contrôlé*. Si le SG A est accepté, l'annotation $\mathbb{A}$ est dite *valide* et dans ce cas elle prend un statut *complet* ou *incomplet* ; sinon elle prend le statut *invalide*. Dans le premier cas, la connaissance produite par l'enrichissement est intégrée

[9] La suppression conserve dans $tm_{\mathcal{A}}$ les éléments de $tm(\mathbb{A})$ qui ne sont pas rigoureusement identiques à $tm_{\mathcal{A}}$ (voir définition de la suppression).

sous forme d'annotations dans $E_{\mathcal{A}}$. Dans les deux cas, les violations des contraintes de $\mathcal{C}^-$ ou $\mathcal{C}^+$ et les réparations sont prises en compte et conservées par le SVE.

L'ajout contrôlé d'une règle ou d'une contrainte, ainsi que le retrait d'une contrainte sont aussi pris en charge. Le résultat de ces opérations est partiellement conservé par le SVE.

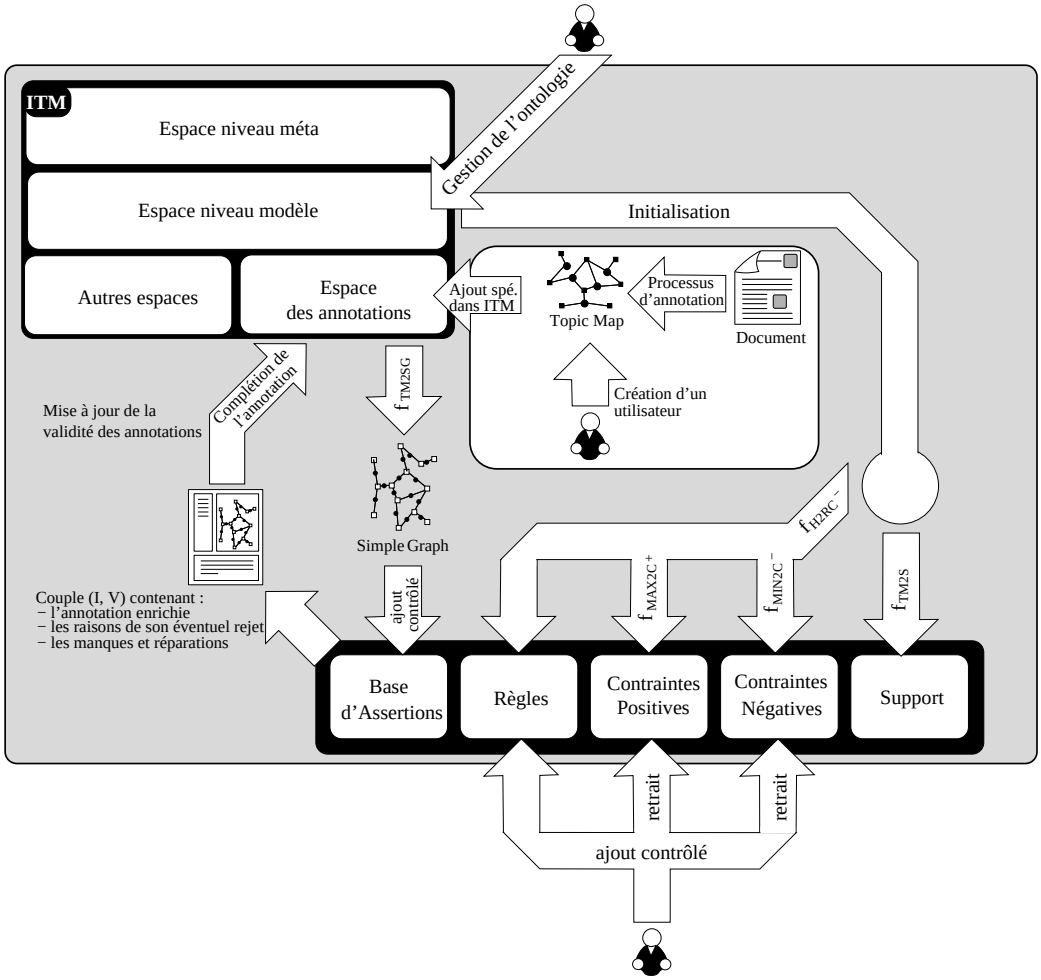


FIG. 5.19 – Architecture du service de validation et d'enrichissement d'ITM.

Le SVE utilise deux fonctions $f_{\mathcal{A}2G}$ et val. La fonction $f_{\mathcal{A}2G}$ permet d'assurer la synchronisation des bases d'annotations TM et SG en établissant une bijection entre la partie *valide* de $tm_{\mathcal{A}}$ et les sommets concepts du graphe G de $B_{\mathcal{SG}}$. La fonction val associe

à chaque annotation les informations concernant sa validité, c'est à dire les contraintes violées et les parties de l'annotation mises en cause.

5.3.1 Validité d'une annotation

La fonction *val* fournit des informations sur la validité d'une annotation. Lorsqu'elle est définie pour un libellé d'annotation donné, elle lui associe un ensemble de couples (C, V_{val}) où C est une contrainte et V_{val} l'ensemble des couples (π, G_C) identifiant les sous-graphes de G violant C après l'ajout contrôlé dans $B_{\mathcal{SG}}$ de l'annotation donnée.

Le statut d'une annotation $\mathbb{A}$ peut être déterminé en fonction du contenu de $val(\mathbb{A})$:

- le statut de $\mathbb{A}$ est *invalide* ssi $val(\mathbb{A})$ contient un couple (C, V) où C est une contrainte négative,
- le statut de $\mathbb{A}$ est *incomplet* (et *valide*) ssi $val(\mathbb{A})$ contient un couple (C, V) où C est une contrainte positive,
- le statut de $\mathbb{A}$ est *complet* (et *valide*) ssi $val(\mathbb{A})$ est vide.

TM valides et invalides de $tm_{\mathcal{A}}$

$tm^{val}_{\mathcal{A}}$ (resp. $tm^{inv}_{\mathcal{A}}$) désigne la plus petite TM de $tm_{\mathcal{A}}$ qui contient toutes les TM associées aux annotations *valides* (resp. *invalides*).

Violations positives et négatives

Si une TM tm incluse dans $tm_{\mathcal{A}}$ est l'image par f_{SG2TM} d'un sous-graphe de G violant une contrainte négative (resp. positive), tm est qualifiée de *violation négative* (resp. *positive*). L'identification des violations négatives (resp. positives) est réalisée de la manière suivante : pour toute annotation $\mathbb{A}$, on détermine les TM de $tm_{\mathcal{A}}$ obtenues par application de f_{SG2TM} sur un SG G_C apparaissant parmi les couples (π, G_C) de V où $(C, V) \in val(\mathbb{A})$ et C est une contrainte négative (resp. positive).

5.3.2 Synchronisation des services

Sous peine d'obtenir des résultats incohérents, les contenus d'ordre assertionnel et ontologique du *SVE* et du service de raisonnement $\mathcal{SG}$ doivent respecter deux conditions dites de *synchronisation*. La première énonce qu'il faut que la portion valide de la base d'annotations du *SVE* reste identique au contenu de la base d'annotation du service de raisonnement $\mathcal{SG}$ (à une transformation près). La seconde déclare que le support et les ensembles de règles/contraintes doivent toujours contenir le résultat de la transformation du niveau modèle.

Condition de synchronisation des bases d'annotations

La condition de synchronisation des bases d'annotations s'appuie sur la fonction $f_{\mathcal{A}2G}$ définie de l'ensemble des topics, des associations, et des valeurs d'occurrences de $tm_{\mathcal{A}}^{val}$ dans l'ensemble des sommets concepts de G. La condition est vérifiée *ssi* la fonction $f_{\mathcal{A}2G}$ est bijective et pour chaque topic, association ou valeur d'occurrence x de $tm_{\mathcal{A}}^{val}$, $f_{\mathcal{A}2G}(x)$ est le sommet concept de G engendré par f_{TM2SG} lors de la traduction de x.

Condition de synchronisation des ontologies

La condition de synchronisation des ontologies est vérifiée ssi :
- le support S de $B_{\mathcal{SG}}$ est le résultat de la traduction par f_{TM2S} de tm_{modele} (condition sur le support) ;
- toute contrainte ou règle produite par l'application sur tm_{modele} des transformations $f_{H2R\mathcal{C}^-}$, $f_{MIN2\mathcal{C}^+}$ et $f_{MAX2\mathcal{C}^-}$ appartient aux ensembles idoines $\mathcal{C}^-$, $\mathcal{C}^+$ et $\mathcal{R}$ (condition sur les règles et contraintes).

Contraintes sur le *SVE*

Ces deux conditions de synchronisation sont vérifiées si le SVE respecte les contraintes suivantes :
- La suppression σ d'une annotation $\mathbb{A}$ dans $tm_{\mathcal{A}}$ n'est permise que si $\mathbb{A}$ est *invalide*. En effet, la suppression d'un sous-graphe de G n'étant pas permise dans $B_{\mathcal{SG}}$, la suppression d'une annotation *valide* entraînerait une désynchronisation car $f_{\mathcal{A}2G}$ ne serait plus bijective ; ce qui violerait la condition de synchronisation entre les bases d'annotations. Le chapitre `Conclusion` décrit une première version de mécanisme de suppression qui conserve les deux bases synchronisées.
- Si l'ajout d'une annotation $\mathbb{A}$ dans $tm_{\mathcal{A}}$ spécialise le type d'un topic et qu'après validation de $\mathbb{A}$ le topic obtenu participe à une *violation négative*, alors l'annotation est rejetée, $tm_{\mathcal{A}}$ n'est pas modifiée par l'ajout spécialisant et la fonction *val* reste inchangée. En effet, un tel ajout désynchroniserait la base car il modifie le type d'un topic dans $tm_{\mathcal{A}}$ sans pour autant modifier celui du sommet concept lui correspondant dans G (l'annotation ayant été rejetée par $B_{\mathcal{SG}}$).
- Toute opération d'ajout ou de suppression est proscrite dans l'espace ontologique tm_{modele}.
- Aucune règle ne peut être supprimée dans $\mathcal{R}$.
- Aucune contrainte obtenue par l'une des transformations ne peut faire l'objet d'un retrait dans $\mathcal{C}^+$ ou $\mathcal{C}^-$.
- En plus des contraintes précédentes qui permettent au SVE de rester synchronisé, il n'est permis de réaliser une disjonction dans la TM $tm_{\mathcal{A}}^{val}$ qu'au cours de l'initialisation de la base (voir partie suivante). D'une part, cela n'est pas utile car on

verra plus loin que cette opération est essentiellement utilisée pour "réparer" des annotations invalides. D'autre part, à la suite d'une telle disjonction il faudrait modifier *val* pour que l'attribution des informations de validité aux annotations modifiées par la disjonction reste cohérente ; ce qui n'a pas été réalisé dans cette version du *SVE*.

S'il est absolument nécessaire de réaliser une action proscrite dans $E_{\mathcal{A}}$ ou dans tm_{modele}, le SVE est réinitialisé après effet de la dite action. La réinitialisation consiste à supprimer l'ensemble des annotations inférées dans $E_{\mathcal{A}}$ et à initialiser le SVE (voir partie suivante), ce qui peut être coûteux en temps si le nombre d'annotations dans la base est important.

5.3.3 Initialisation

A l'initialisation du service le support S, l'ensemble $\mathcal{R}$ de règles et les ensembles de contraintes $\mathcal{C}^+$ et $\mathcal{C}^-$ de $B_{\mathcal{SG}}$ sont obtenus en appliquant les transformations f_{TM2S}, f_{H2RC^-}, f_{MAX2C^-} et f_{MIN2C^+} sur tm^{modele}.

Types particuliers de règles et de contraintes

Les ensembles $\mathcal{R}$, $\mathcal{C}^+$ et $\mathcal{C}^-$ de règles, de contraintes négatives et positives peuvent être manuellement complétés en utilisant un outil d'édition de règles et contraintes $\mathcal{SG}$ (l'outil CoGUI, voir [Gut06]). Certaines règles et contraintes de la famille $\mathcal{SG}$ sont réservées à un usage précis. Celles qui conduisent à spécialiser ou vérifier le type d'un sommet appartiennent à la catégorie des règles ou contraintes de *spécialisation de type*. Les contraintes définies pour vérifier qu'il n'existe pas de sommets différents qui désignent une même entité appartiennent à la catégorie des *contraintes d'identification*.

Règles et contraintes positives de spécialisation de type

Une règle R (resp. une contrainte positive C) de spécialisation de type présente un lien de coréférence entre un sommet concept frontière s de type t et un sommet concept non frontière s' de type $t' < t$ de la partie conclusion (resp. obligation). Soit une projection π de l'hypothèse de R dans G, l'application d'une telle règle spécialise le sommet $\pi(s)$ de telle sorte que le nouveau sommet obtenu soit de type t' ssi $t' \leq type(\pi(s))$. La contrainte positive C de spécialisation est dite violée ssi il existe une projection π de la partie condition de la contrainte telle que le sommet $\pi(s)$ soit de type t sans être d'un sous-type de t'.

Il faut remarquer que les règles où contraintes positives qui présentent un lien de coréférence entre deux sommets de types t et t' incomparables sont rejetées par $B_{\mathcal{SG}}$. Pour parvenir à appliquer une telle règle où à satisfaire une telle contrainte il faudrait

attribuer au sommet concerné un type dit *conjonctif*. Les types conjonctifs sont définis dans la famille $\mathcal{SG}$, mais ne sont pas encore gérés par $B_{\mathcal{SG}}$.

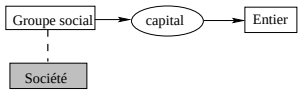


FIG. 5.20 – La règle R_3 de spécialisation de type.

La figure 5.20 présente la règle R_3 de spécialisation de type selon laquelle "un groupe social ayant un capital est une société". Comme les contraintes positives sont aussi bicolorées, cette règle peut être vue comme une contrainte positive. Dans ce cas, la contrainte signifie "qu'un groupe social ayant un capital doit être une société".

Contraintes d'identification

Une contrainte d'identification est une contrainte négative qui présente un lien de *différence* (relation $\neq$) entre deux sommets concepts s et s'. Une telle contrainte est violée par le graphe G lorsqu'il existe une projection de cette contrainte dans G qui projette s sur une image différente de celle de s'.

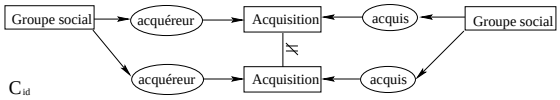


FIG. 5.21 – Une contrainte d'identification.

La figure 5.21 présente une contrainte négative signifiant que deux associations d'acquisition dans lesquelles interviennent deux mêmes topics - chacun jouant dans l'une un rôle identique à celui joué dans l'autre - ne peuvent être différentes.

Préparation de la base d'annotation

A l'initialisation, $tm_{\mathcal{A}}$ peut ne pas être vide et dans ce cas :

(1) soit la base est divisée en annotations de taille convenable [10] (chaque sommet de $tm_{\mathcal{A}}$ a une image par *annotations*) ;

(2) soit $tm_{\mathcal{A}}$ ne contient aucune annotation (la fonction *annotations* est indéfinie). Cela se produit généralement lorsque les connaissances représentées dans $tm_{\mathcal{A}}$ proviennent d'une TM standard ; la notion de métadonnée dans ITM étant externe au

[10] La taille d'une annotation est jugée convenable si elle ne dépasse pas une certaine borne déterminée empiriquement.

standard XTM. Dans ce cas, la TM $tm_{\mathcal{A}}$ n'est pas initialement découpée sous forme d'annotations. La solution consiste à utiliser la disjonction δ pour la morceler en autant d'annotations *disjointes* ou *adjacentes* de taille bornée : pour un entier n donné, on réalise la disjonction δ d'une TM connexe de taille n dans $tm_{\mathcal{A}}$. Récursivement, on applique δ sur de nouvelles TM de taille inférieure où égale à n qui sont *adjacentes* aux TM précédemment obtenues par disjonction ; et ceci jusqu'à ce que tout élément de $tm_{\mathcal{A}}$ appartienne à une annotation. Pour chaque disjonction, on utilise un nouveau libellé $\mathbb{A} \in L_{\mathcal{A}}$ différent de ceux qui identifient déjà une annotation.

Une fois la base prête, les annotations peuvent être ajoutées par des utilisateurs ou des agents logiciels.

5.3.4 Services de raisonnement pour ITM

Le *SVE* d'ITM autorise quatre actions : (1) la validation et l'enrichissement d'une annotation de l'espace $E_{\mathcal{A}}$, (2) le retrait d'une contrainte des ensembles $\mathcal{C}^-$ et $\mathcal{C}^+$ et (3) l'ajout contrôlé d'une règle ou (4) d'une contrainte aux ensembles $\mathcal{R}$, $\mathcal{C}^-$ et $\mathcal{C}^+$.

Validation et enrichissement d'une annotation

La *validation et l'enrichissement* d'une annotation $\mathbb{A}$ d'ITM est réalisée en deux étapes : la première consiste à traduire $tm(\mathbb{A})$ en un *SG* et à réaliser un ajout contrôlé de ce *SG* en utilisant le service de raisonnement $\mathcal{SG}$; et la seconde à traduire le résultat retourné par le service $\mathcal{SG}$ pour qu'il soit exploitable par ITM.

La procédure de validation et d'enrichissement d'une annotation $\mathbb{A}$ est déclenchée si :

(1) $\mathbb{A}$ est une nouvelle annotation issue d'un ajout spécialisant α d'une TM tm dans $tm_{\mathcal{A}}$ ou d'une disjonction δ dans $tm_{\mathcal{A}}^{inv}$;

(2) $\mathbb{A}$ est *adjacente* à une nouvelle annotation issue d'une disjonction δ dans $tm_{\mathcal{A}}^{inv}$;

(3) $\mathbb{A}$ est *adjacente* à une annotation $\mathbb{A}'$ supprimée par un suppression généralisante σ ;

(1) Tout d'abord, le *SVE* réalise la traduction de tm en un *SG* X en utilisant f_{TM2SG}. La fonction i_X associant à chaque sommet concept de X un identifiant de $\mathcal{L}_G$ est obtenue[11] à partir de $f_{\mathcal{A}2G}$. La figure 5.22 présente un exemple d'annotation et de sa traduction en SG.

[11] Si $f_{\mathcal{A}2G}(x)$ existe, $i_X(c) = i(f_{\mathcal{A}2G}(x))$ où i est la fonction d'identification associée à la base G de $B_{\mathcal{SG}}$, x est un topic, une association où une valeur d'occurrence et c est le sommet concept engendré par f_{TM2SG} lors de la traduction de x. Sinon, $i_X(c)$ est un nouveau libellé qui n'a pas d'antécédent par i hormis x.

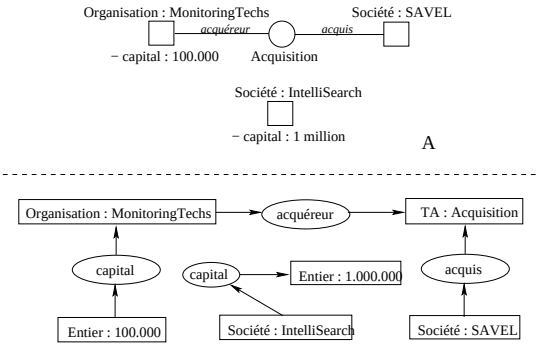


FIG. 5.22 – L'exemple d'une annotation et de sa traduction en SG.

(2) Ensuite, le *SVE* effectue l'ajout contrôlé de (X, i_X) à $B_{\mathcal{SG}}$, puis intègre dans ITM le résultat $(X_I, \mathbb{V})$ retourné. La procédure d'intégration est détaillée en section 5.3.4.

Lorsque les SG résultant de la traduction des annotations à valider et à enrichir sont assurés d'être sous forme normale ou de respecter certaines conditions parmi les quatre énumérées dans la partie détaillant la procédure de contrôle (voir l'étape 1 de l'ajout contrôlé décrite à la section 5.1.3), la vérification de ces conditions ou la mise sous forme normale est désactivée dans $B_{\mathcal{SG}}$; cela permet d'accélérer le traitement.

Ajout d'une règle ou d'une contrainte

Après l'ajout contrôlé d'une règle dans $\mathcal{R}$ ou d'une contrainte dans $\mathcal{C}^-$ ou $\mathcal{C}^+$, le *SVE* intègre dans ITM le résultat $(I, \mathbb{V})$ retourné par $B_{\mathcal{SG}}$ si $\mathbb{V}$ ne contient aucune violation de contraintes négatives (voir section 5.3.4).

Retrait d'une contrainte

Le retrait d'une contrainte n'est possible que si elle ne conduit pas à l'élimination d'une contrainte de $\mathcal{C}^+$ ou $\mathcal{C}^-$ issue de l'application de f_{H2RC^-}, f_{MAX2C^-} et f_{MIN2C^+} sur tm_{modele}. Le *SVE* intègre ensuite dans ITM le résultat $(\emptyset, \mathbb{V})$ retourné par $B_{\mathcal{SG}}$.

Intégration du résultat retourné par $B_{\mathcal{SG}}$

Dans la suite, $(X_I, \mathbb{V})$ désigne le résultat retourné par $B_{\mathcal{SG}}$ après l'ajout contrôlé d'une annotation $\mathbb{A}$ ou d'une règle R. X_I est le sous-graphe de G induit par les sommets ajoutés et inférés au cours de l'ajout contrôlé.

L'ensemble $\mathbb{V}$ contient les informations concernant la validité de l'annotation ajoutée. Il contient des couples (C, V) où C est une contrainte négative, et des triplets (C, V, R)

où C est une contrainte positive. L'ensemble V des couples ou des triplets contient des informations concernant les "violations" de la contrainte C qu'elle soit négative ou positive et l'ensemble R des triplets contient des informations concernant les "réparations" de la contrainte C qui dans ce cas ne peut être que positive. Plus précisément, V et R sont deux ensembles contenant des couples (π, G_C) où G_C est l'image par π de la contrainte C entière si elle est négative ou de sa condition si elle est positive.

Si $\mathbb{V}$ ne contient aucun couple (C, V), cela signifie qu'aucune contrainte négative a été violée. Dans ce cas, on met à jour l'espace des annotations *et* la fonction *val*. Dans le cas contraire, on *ne* met à jour *que* la fonction *val*.

Mise à jour de l'espace des annotations

Si X_I est non vide, l'application de f_{SG2TM} sur X_I fournit la TM tm_{X_I}. On réalise ensuite l'ajout spécialisant α de tm_{X_I} en tant que nouvelle annotation $\mathbb{A}_{X_I}$ de $tm_{\mathcal{A}}$. Cette étape fusionne (et spécialise) dans $tm_{\mathcal{A}}$ les éléments qui se trouvent dans tm_{X_I}. Les éléments de tm_{X_I} qui ont été produits par inférence ou dont le type a été spécialisé sont différenciés[12] de ceux initialement présents dans l'annotation envoyée.

La bijection $f_{\mathcal{A}2G}$ est maintenue à jour. Chaque topic, association ou valeur d'occurrence créé dans $tm_{\mathcal{A}}$ et mis en bijection par $f_{\mathcal{A}2G}$ avec le sommet concept lui correspondant dans G.

Les figures 5.23, 5.24 et 5.25 présentent respectivement l'état de l'espace des annotations : la première illustrant l'état initial de la base d'annotations, la seconde après enrichissement et validation de l'annotation A (en noir et en gras) et la troisième après l'intégration de l'annotation rejetée A^R (en noir) à la suite de celle de A. Dans la seconde figure, la classe `Organisation` du topic `MonitoringTechs` a été spécialisée en `Société` par l'enrichissement et les sommets inférés sont entourés de gris. Cette figure présente une violation négative entourée de tirets et une violation positive située dans la portion hachurée. Dans la troisième figure, l'annotation en noir ayant été rejetée les inférences n'ont pas été intégrées à l'espace des annotations.

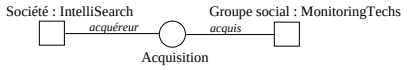


FIG. 5.23 – Le contenu de la base d'annotations avant intégration des deux annotations.

[12] Une métadonnée *source* de l'espace $E_{\mathcal{A}}$ attribue à chaque élément de $tm_{\mathcal{A}}$ un libellé identifiant l'agent qui l'a créée. Dans ce cas-ci, tout élément x est considéré *inféré* si $source(x) = SVE$.

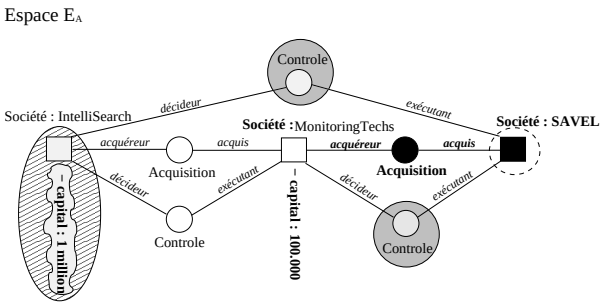


FIG. 5.24 – Le contenu résultant de l'intégration dans l'espace des annotations de l'annotation enrichie et validée retournée par $B_{\mathcal{SG}}$.

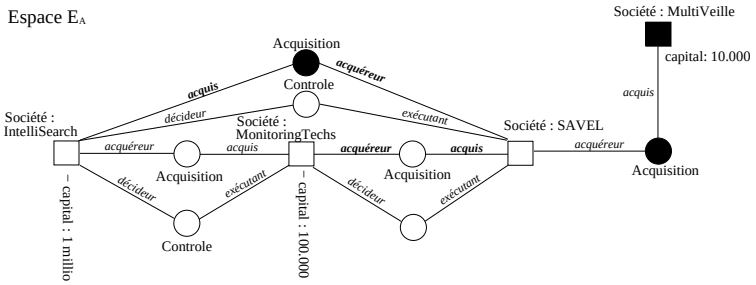


FIG. 5.25 – Le contenu résultant de l'intégration d'une annotation rejetée par $B_{\mathcal{SG}}$.

Mise à jour des informations de validité

Le *SVE* utilise les informations de $\mathbb{V}$ pour mettre à jour le contenu de $val(\mathbb{A})$.
Pour chaque libellé d'annotations $\mathbb{A}$ et pour chaque couple (C, V_{val}) de $val(\mathbb{A})$,
- si C est positive, on considère s'il existe le triplet (C, V, R) de $\mathbb{V}$, on supprime de V_{val} les couples (π, G_C) qui se trouvent dans R ; et on ajoute à V_{val} les couples (π, G_C) qui se trouvent dans V ;
- si C est négative, on considère s'il existe le couple (C, V) de $\mathbb{V}$ et on ajoute à V_{val} les couples (π, G_C) qui se trouvent dans V.

Pour le moment le *SVE* ne conserve pas les informations retournées par un ajout contrôlé d'une règle ou d'une contrainte négative qui se solde par un échec. Les problèmes liés à leur conservation constituent néanmoins des perspectives de recherches qui seront abordées prochainement afin d'améliorer la gestion et la compréhension des violations.

L'espace des annotations de la figure 5.24 présente une violation positive entourée de

pointillés et une réparation de violation positive (partie hachurée) ; toutes deux dues à l'ajout de l'annotation coloriée en noir dans la base illustrée à la figure 5.23. La partie hachurée violait la contrainte positive de la figure 5.26 qui impose à toute société d'avoir un capital. Cette même contrainte est violée par l'annotation coloriée en noir (la violation positive étant encerclée de tirets).

Affichage des résultats de la validation et de l'enrichissement

Un écran divisé en trois parties, et consultable à tout moment, présente le résultat de la validation et de l'enrichissement des annotations. Chacune des trois parties dresse la liste des libellés des annotations respectivement *complètes*, *incomplètes* et *invalides*. Lorsqu'on sélectionne un libellé $\mathbb{A}$, la TM $tm(\mathbb{A})$ est affichée sous une forme graphique (les topics sont des carrés, les associations des disques et les arêtes des segments). Si l'annotation est *valide* le contenu inféré de $tm(\mathbb{A})$ est représenté à l'écran d'une couleur différente de celle utilisée pour le contenu asserté.

Lors de l'affichage d'une annotation $\mathbb{A}$, l'écran qui apparaît dresse en plus un inventaire des contraintes C violées par l'annotation choisie : en d'autres termes, on présente les contraintes C de chaque couple $(C, V) \in val(\mathbb{A})$. Pour les besoins de l'affichage, chaque contrainte C est identifiée par un libellé et un commentaire décrit la contrainte. Lorsqu'on sélectionne le libellé d'une contrainte C, une nouvelle fenêtre affiche les *violations* de la contrainte sélectionnée. Plus précisément, on considère le couple $(C, V) \in val(\mathbb{A})$ où C est la contrainte sélectionnée et on affiche à l'écran sous la forme de graphes les TM tm obtenues par application de f_{SG2TM} sur les SG G_C de chaque couple (π, G_C) de V. Cet affichage différencie comme le précédent les sommets inférés des sommets assertés.

Le contenu provenant de l'enrichissement des annotations ayant été intégré à la base (lorsque l'annotation est *valide*), il peut aussi être consulté à l'aide des moyens standards mis à disposition par ITM, à savoir : les interfaces de recherches et l'interface de navigation.

5.3.5 Maintenance des annotations selon leur statut

La maintenance est réalisée en réparant, ou en améliorant la base d'annotations. La *réparation* et l'*amélioration* de $E_{\mathcal{A}}$ consistent à diminuer le nombre d'annotations respectivement *invalides* et *incomplètes* dans $E_{\mathcal{A}}$.

Amélioration d'une annotation *incomplète*

Une annotation éventuellement enrichie est *incomplète* lorsque ses éléments ne sont pas assez spécialisés ou qu'il lui en manque pour être considérée *complète*.

L'amélioration d'une annotation $\mathbb{A}$ *incomplète* consiste à diminuer le nombre n de contraintes positives violées par $\mathbb{A}$ en réalisant un ajout spécialisant qui va compléter ou

spécialiser au moins une *violation positive* de $\mathbb{A}$ afin d'obtenir une nouvelle annotation qui viole un nombre de contraintes positives strictement inférieur à n.

Si la suppression généralisante dans une TM *valide* n'avait pas été proscrite, l'amélioration d'une annotation *incomplète* aurait aussi pu être réalisée en supprimant la *violation positive* à l'origine du manque de connaissance. En effet, une violation positive correspond à la traduction en TM de l'image par projection π dans G de la partie *condition* d'une contrainte positive. Si la violation positive avait pu être supprimée, la partie *condition* ne s'y serait plus projetée selon π et la contrainte n'aurait plus été violée par cette portion de la base.

La figure 5.27 présente l'espace des annotations après l'ajout d'une annotation A_{Am} dans le but d'améliorer l'annotation incomplète présentée aux figures 5.24 et 5.25 (topic `SAVEL`). La portion entourée de pointillés correspond à l'annotation A_{Am} après intégration. L'amélioration réside en ce que le sommet `SAVEL` ne viole plus la contrainte positive de la figure 5.26 qui impose à toute société d'avoir un capital.

FIG. 5.26 – Contrainte positive qui impose à toute société d'avoir un capital.

Réparation d'une annotation *invalide*

La réparation d'une annotation $\mathbb{A}$ *invalide* consiste à diminuer le nombre n de contraintes négatives violées par $\mathbb{A}$ en réalisant : une disjonction dans $tm(\mathbb{A})$ ou bien une suppression généralisante (pouvant généraliser le type de certains topics de $\mathbb{A}$ sans pour autant les supprimer) afin d'obtenir une nouvelle annotation qui viole un nombre de contraintes négatives strictement inférieur à n.

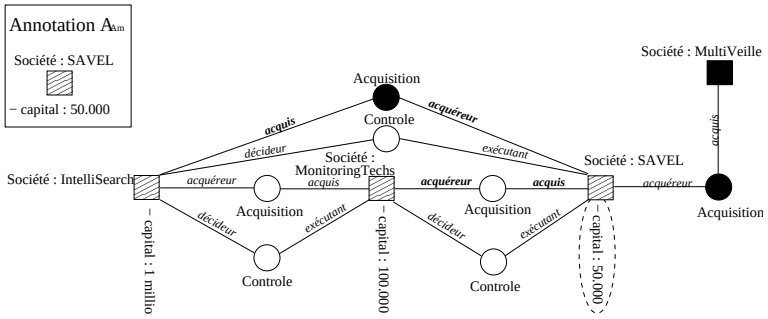


FIG. 5.27 – L'espace des annotations et des violations résultant de l'amélioration d'une annotation incomplète.

Une contrainte d'identification est violée lorsque des éléments vérifiant certaines propriétés sont différents alors qu'ils ne devraient pas l'être. L'annotation peut être réparée en créant par disjonction une nouvelle annotation identique à l'annotation $\mathbb{A}$ à ceci près qu'elle exclut un de ces éléments.

La figure 5.29 présente l'espace des annotations après (1) la disjonction de la TM induite par l'association d'acquisition et les topics `SAVEL` et `IntelliSearch` sous la forme d'une nouvelle annotation (2) puis sa suppression de l'espace. La disjonction répare l'annotation coloriée en noir qui est acceptée par $B_{\mathcal{SG}}$ puisqu'elle ne viole plus la contrainte de la figure 5.28 qui interdit à l'association de type `Contrôle` de former des boucles.

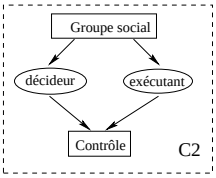


FIG. 5.28 – Contrainte négative qui interdit à l'association de type `Contrôle` de former des boucles.

Il se peut qu'une contrainte négative soit violée par le contenu produit lors de l'enrichissement de l'annotation. Cette situation peut être due à la forme des règles de l'ensemble $\mathcal{R}$ ou au contenu du sous-graphe de G à partir duquel les inférences ont démarré. Ce cas est relativement problématique car le contenu inféré n'étant pas intégré dans $tm_{\mathcal{A}}$, il ne peut être consulté à partir de l'interface de navigation. En effet, des violations appartenant à un contenu inféré ne peuvent être consultées qu'à partir de l'écran de consultation des résultats de la validation. L'affichage de ces violations n'étant pas très explicite, l'utilisateur doit lui même rechercher les causes d'une telle violation pour déterminer si elle provient du jeu de règles ou de l'annotation.

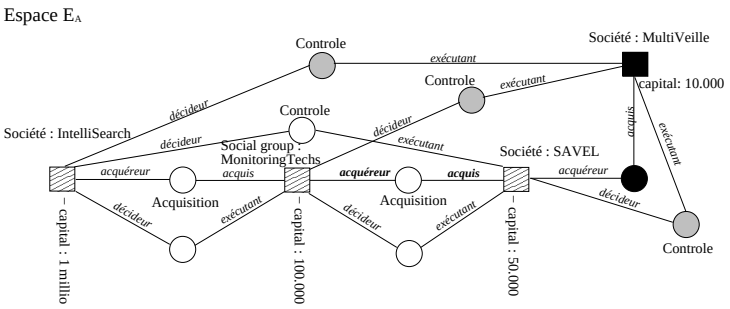


FIG. 5.29 – L'espace des annotations et des violations après réparation d'une annotation rejetée.

5.3.6 Cas d'applications

Le service de validation et d'enrichissement d'ITM est utilisé dans deux projets : "PressIndex" et "Eiffel". Le premier projet a conduit à la réalisation d'une application dédiée à la veille médiatique et le second, en cours, consiste à développer un service d'accès aux ressources touristiques du territoire tout en étant adapté aux besoins promotionnels des organisations nationales du tourisme.

Le projet PressIndex

La veille médiatique est un service très prisé dans les secteurs politiques, industriels ou commerciaux. Elle consiste à synthétiser et discriminer, selon la pertinence, les sources et le contenu, des informations provenant de divers médias dans un secteur d'actualité précis, facilitant ainsi la prise de décision concernant ce secteur.

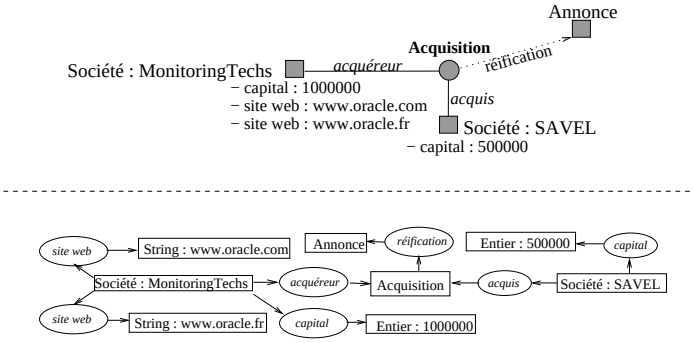


FIG. 5.30 – Un exemple d'annotation du projet PressIndex et de sa traduction en SG.

Dans le cas du projet PressIndex, il s'agit plus précisément d'une veille concurrentielle ciblant les secteurs du commerce et de l'économie afin de récolter des informations sur l'environnement concurrentiel des sociétés. Elle porte sur plus de 9500 sources médiatiques. L'objectif est d'automatiser deux tâches du travail du veilleur. La première consiste à identifier les faits, les rumeurs et les annonces officielles extraits des dépêches provenant des grandes agences de presse (telles que l'AFP, Reuters, etc). La seconde consiste à évaluer la pertinence des informations extraites en repérant des confirmations, des infirmations ou des contradictions de rumeurs et d'annonces. Les indications obtenues permettent ensuite d'analyser à partir des faits l'environnement concurrentiel d'une société ; ou d'anticiper son évolution à partir des annonces et des rumeurs.

L'outil ITM répond à ces besoins grâce à son infrastructure de stockage massif de différents types de connaissances et son service d'annotation automatique de documents basé sur des techniques de traitement automatique du langage naturel. L'application PressIndex de veille médiatique est composée du service de validation et d'enrichissement d'ITM où l'espace des annotations $E_{\mathcal{A}}$ est alimenté par le service d'annotation automatique ([Ama06]).

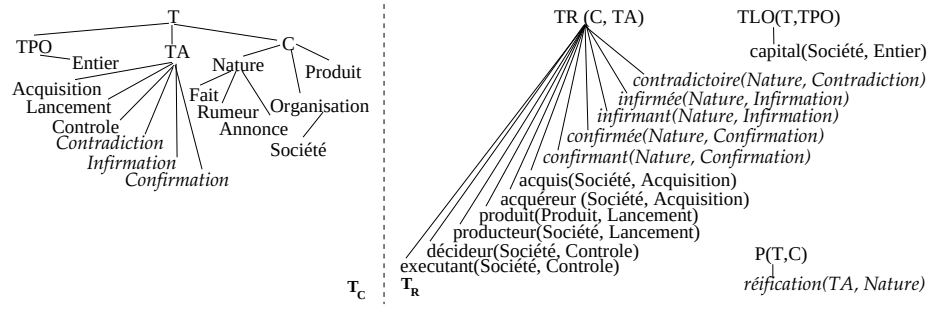


FIG. 5.31 – Une partie du support obtenu à partir du niveau modèle du projet PressIndex.

Le service de raisonnement $B_{\mathcal{SG}}$ est passablement différent de la version présentée dans la première section de ce chapitre. Il offre deux services de raisonnement : (1) le service d'ajout contrôlé et (2) un service d'évaluation de la pertinence des annotations construites. $B_{\mathcal{SG}}$ est initialisé à partir des ensembles de règles $\mathcal{R}_A$ et de contraintes ${C_A}^+$ et ${C_A}^-$ en partie issus des transformations f_{H2RC^-}, f_{MIN2C^+} et f_{MAX2C^-} ; et le support S à partir duquel est défini sa base est issu de la transformation f_{TM2S}.

La première tâche du veilleur qui consiste à identifier les faits, les annonces ou les rumeurs dans les dépêches est réalisée par le processus d'annotation. On lui fournit des dépêches portant sur l'actualité économique et ce dernier en extrait les principaux événements dont il détermine la nature avant de les modéliser en TM sous forme d'annotations. La figure 5.30 présente une annotation de ce type.

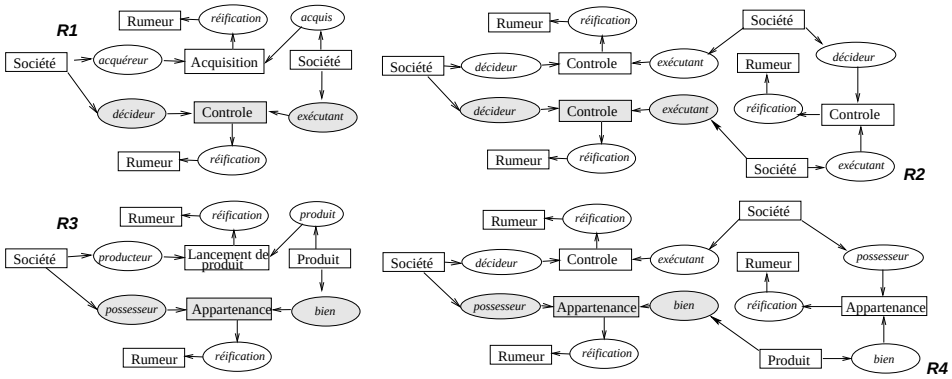


FIG. 5.32 – Les règles de $\mathcal{R}_A$ associées au vocabulaire d'annotation.

Le service d'annotation utilise une "cartouche linguistique" qui établit les motifs syntaxiques intéressants à repérer. Cette cartouche a été conçue par une société spécialisée à partir de l'expertise des différents types de dépêches relevant du secteur veillé. Les motifs syntaxiques de la cartouche sont définis conformément à un vocabulaire structuré qu'il est nécessaire de mettre en correspondance avec celui d'ITM pour pouvoir représenter dans la base les informations extraites. Pour chaque dépêche, le service d'annotation produit une TM et attribue à chaque association une nature *rumeur*, *annonce* ou *fait* en lui affectant un `pointeur` vers le topic indiquant sa nature. Par exemple, la TM de la figure 5.30 présente une association `Acquisition` dont la nature est *annonce*.

Le service d'ajout contrôlé (voir partie 5.1.3) permet de contrôler la cohérence des faits et de déduire de nouvelles connaissances à partir de celles extraites par le service d'annotation.

La seconde tâche du veilleur qui consiste à détecter des confirmations, infirmations ou contradictions de rumeurs et d'annonces est accomplie par le service d'évaluation de la pertinence. L'annotation enrichie par le service précédent est ensuite transmise à ce service qui en déduit des relations entre faits, annonces ou rumeurs susceptibles de déclencher des alertes de veille. Il s'agit plus précisément de détecter des contradictions, confirmations ou infirmations de rumeurs et d'annonces. Ce service est un service d'ajout contrôlé (voir partie 5.1.2) qui partage le support et la base d'assertions du premier service d'ajout contrôlé.

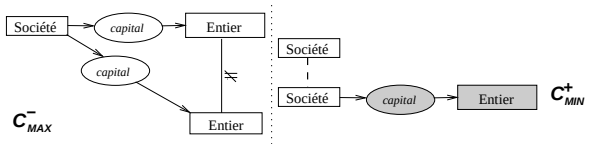


FIG. 5.33 – Les contraintes de $C_A{}^+$ et $C_A{}^-$ associées au vocabulaire d'annotation.

L'ensemble des règles qui lui est assigné est noté $\mathcal{R}_P$ et ses ensembles de contraintes sont vides. Le support à partir duquel est définie la base des deux services d'ajout contrôlé est divisé en deux parties :

- le *vocabulaire d'annotation* : il regroupe les types d'entités et d'événements du domaine veillé à partir desquels sont définies les annotations traduites en SG. Dans le support de la figure 5.31 les types de ce vocabulaire ne sont pas en italique.
- le *vocabulaire d'évaluation de la pertinence* : il définit de manière permanente des relations (confirmation, infirmation, contradiction) permettant de qualifier la pertinence des connaissances extraites. Dans la figure 5.31, il est représenté en italique.

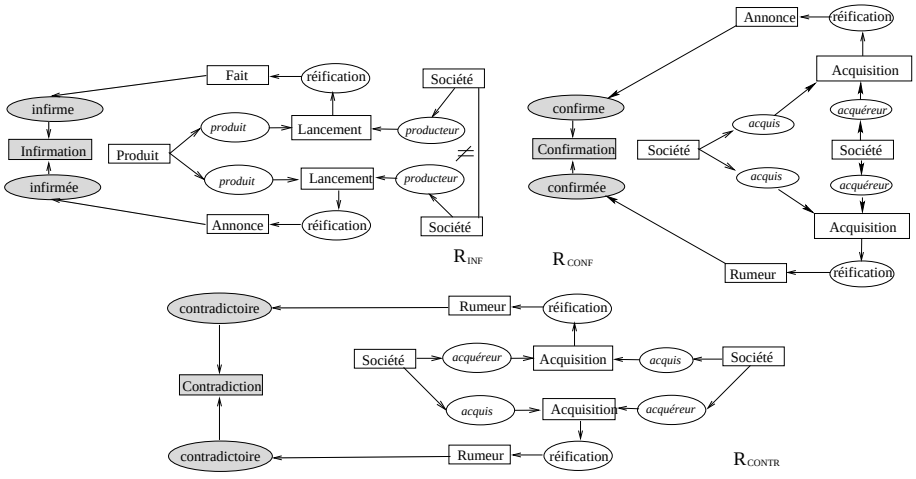


FIG. 5.34 – Une règle infirmant une annonce à partir d'un fait, une règle qui confirme une rumeur à partir d'une annonce et une règle de détection de contradiction entre rumeurs.

Les ensembles de règles $\mathcal{R}_A$ et de contraintes $\mathcal{C}_A{}^+$ et $\mathcal{C}_A{}^-$ sont associés au vocabulaire d'annotation. Les conclusions des règles et les contraintes dont un exemple est présenté aux figures 5.32 et 5.33 ne sont définies qu'avec les types du vocabulaire d'annotation. Ces ensembles sont obtenus à partir des fonctions f_{H2RC^-}, f_{MAX2C^-} et f_{MIN2C^+}, et peuvent

être complétés par un expert du domaine via l'interface CoGUI.

L'ensemble $\mathcal{R}_P$ des règles d'évaluation de la pertinence est associé au vocabulaire de la pertinence. Il rassemble deux genres de règles différentes manuellement crées par un expert du domaine : (1) les règles assurant la détection de contradictions entre annonces ou entre rumeurs et (2) celles permettant de confirmer/infirmer des rumeurs ou des annonces à partir des faits. La conclusion de ces règles est exclusivement formée de sommets dont les types appartiennent au *vocabulaire de la pertinence*. Ces règles ne peuvent donc pas créer de violations parmi les contraintes de ${C_A}^-$ et ${C_A}^+$.

La détection de contradictions et d'infirmations est réalisée par inférence et non par vérification de contraintes négatives car des configurations singulières qui n'en sont pas pour autant absurdes, telles que "l'infirmation par un fait d'une annonce qui confirmait une rumeur" peuvent intéresser le veilleur si, par exemple, il cherche à déterminer le plus rapidement possible l'événement qui créé la surprise, et permet de prendre à contre-pied les concurrents qui s'attendaient fortement à ce que se produise celui décrit dans la rumeur appuyée par l'annonce. Or, utiliser la violation d'une contrainte négative pour détecter une telle configuration aboutit à un rejet de l'annotation (voir 5.1.2) qui prive le veilleur de ces indications et supprime de la base des connaissances qui auraient pu conduire à de nouvelles inférences.

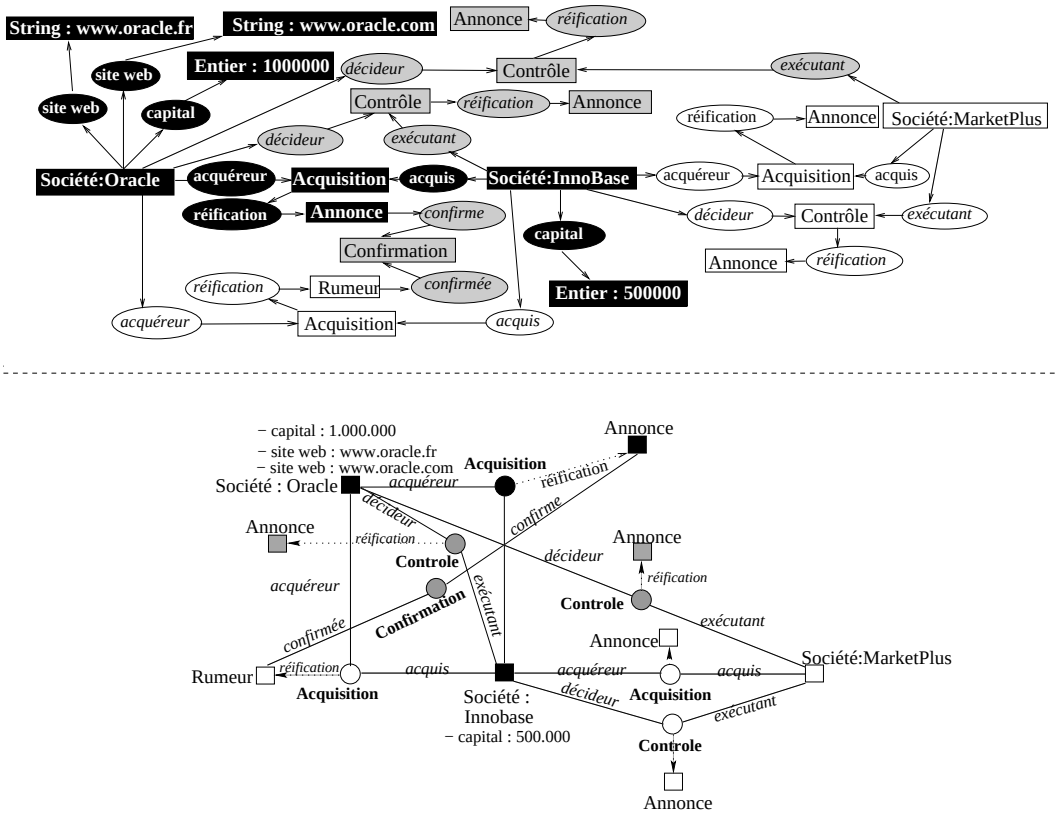


FIG. 5.35 – Un exemple de base d'annotations ITM et d'assertions SG après l'ajout contrôlé de l'annotation de la figure 5.30.

Formalisées en SG, les règles de détection de contradictions entre annonces (resp. rumeurs) produisent un sommet concept de type `Contradiction` relié par deux relations `contradictoires` à deux concepts réifiant des annonces (resp. rumeurs). Par exemple, l'existence d'une acquisition de la société A par la société B et d'une acquisition opposée de B par A est considérée comme contradictoire et déclenche ce type de règle.

Une annonce ou rumeur peut être confirmée (resp. infirmée) par un fait. Pour cela il faut que les deux portent sur le même type d'association et concernent les mêmes entités. Formalisées en SG, la conclusion de telles règles produit un sommet concept de type `Confirmation` (resp. `Infirmation`) relié par deux relations `confirmant` (resp. `infirmant`) `confirmée` (resp. `infirmée`) à un concept de type `Annonce` (ou `Rumeur`) et un autre de type `Fait`.

En plus d'être confirmée (resp. infirmée) par un fait, une rumeur peut l'être par une annonce. Dans ce cas le sommet concept confirmant (resp. infirmant) est de type `Annonce`.

La figure 5.34 montre un exemple (1) de règle de détection d'infirmation de l'annonce du lancement d'un produit par une société, par le fait que ce produit est mis en vente par une autre société, (2) d'une règle de confirmation d'une rumeur d'acquisition de société par une annonce équivalente et (3) d'une règle de détection de contradiction entre deux rumeurs d'acquisition de société.

L'ensemble de l'application fonctionne de la manière suivante : Lorsque le service d'annotation créé une nouvelle annotation, cette dernière est transférée au service de validation et d'enrichissement qui valide et enrichit l'annotation dans sa base. Dans le cas où elle n'est pas rejetée, la connaissance inférée est retournée à ITM. L'annotation enrichie est ensuite transférée au service d'évaluation de la pertinence qui ne réalise pas l'étape d'intégration et infère les associations de contradiction, confirmation et infirmation directement sur l'annotation dans sa base avant de les fournir à ITM. Dans le cas où l'annotation est rejetée par le service de validation et d'enrichissement, elle n'est pas transmise au service d'évaluation de la pertinence. La réception par ITM des associations inférées sur la pertinence ou des rapports sur la violation des contraintes lève des alertes de différentes natures exploitables par l'utilisateur.

La figure 5.35 présente l'état des bases après ajout de l'annotation de la figure 5.30. L'annotation inférée présentée en gris et celle ajoutée en noir correspondent au contenu retourné par $B_{\mathcal{SG}}$ après enrichissement, validation et évaluation de la pertinence. L'association `Confirmation` a été inférée au cours de l'évaluation de la pertinence et le reste de la partie grise au cours de l'étape d'enrichissement.

Le projet Eiffel

Une organisation nationale du tourisme (ONT) est un organisme public destiné à promouvoir et développer le tourisme d'un pays. Pour décrire des ressources touristiques, elle doit s'aligner sur des standards et des normes. En France, par exemple, l'ONT doit respecter le format TourinFrance [Tou]. Ce format propose treize types de ressources touristiques (hotels, restaurants, ressources naturelles, activités, etc) pour décrire l'ensemble des offres du territoire français. Pour mieux promouvoir ces offres touristiques, le territoire doit être décrit (1) de manière à répondre aux attentes de la demande selon des stratégies promotionnelles ; (2) de façon à fournir des méthodes d'accès rapides et intuitives (au lieu de formuler des requêtes précises, l'utilisateur interroge le système avec des termes plus vagues et subjectifs) et (3) en caractérisant une ressource touristique relativement aux autres pour faciliter la recherche et favoriser la découverte de nouveaux sites (il peut être intéressant d'attirer l'utilisateur vers un site de petite notoriété en l'associant avec un site célèbre proposant des activités similaires).

Le projet Eiffel a pour but de définir un service qui réponde à la fois aux besoins de l'utilisateur et à ceux de l'ONT. Ce service doit fournir des possibilités d'intégration et d'enrichissement de ressources touristiques, de mise en relations de ces ressources

et de restitution des informations. Il est conçu à partir d'un système de gestion des connaissances basé sur ITM et permet aux experts du tourisme d'organiser, de modifier et d'enrichir manuellement les ressources touristiques d'un territoire. Ces dernières étant en nombre considérable, leur représentation dans la base et leur maintenance ne sont pas toujours manuellement réalisables. Le projet Eiffel vise à automatiser certaines tâches de l'expert et proposer à l'utilisateur un service évolué de recherche d'offres.

L'application fonctionne en deux phases : une phase d'acquisition et une phase de diffusion. La figure 5.36 présente le fonctionnement global de l'application.

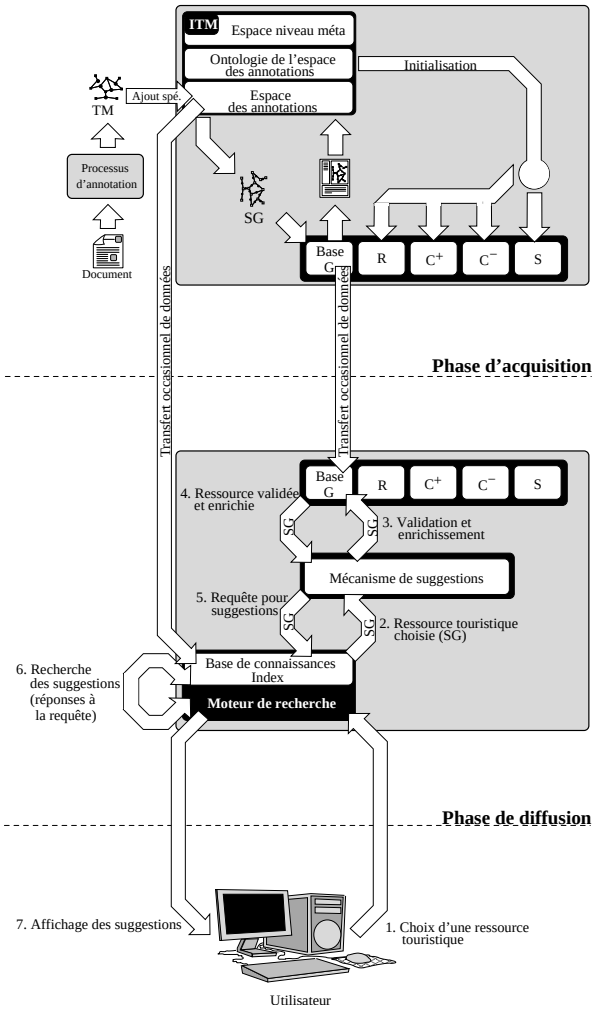


FIG. 5.36 – Architecture globale du projet Eiffel.

La première phase consiste à acquérir de nouvelles ressources touristiques. L'acquisition peut être réalisée manuellement ou semi-automatiquement en appliquant des méthodes d'extraction linguistique sur le contenu textuel présentant les ressources touristiques. Les annotations représentant ces ressources sont ensuite enrichies dans la base et leur validité est contrôlée. La seconde phase consiste à présenter ces offres de manière intuitive pour l'utilisateur en se basant sur leurs propriétés sémantiques et proposer automatiquement à l'utilisateur des suggestions potentiellement intéressantes pour lui.

La première phase est réalisée par un service de validation et d'enrichissement où l'espace des annotations $E_{\mathcal{A}}$ contient des connaissances de géolocalisation (par exemple, une ville et son pays) et est alimenté par un service d'annotation automatique ([Ama06]) spécialisé dans la recherche et la modélisation en TM d'offres touristiques.

Les règles et les contraintes décrivent essentiellement la sémantique du vocabulaire de géolocalisation. La figure 5.37 présente la règle selon laquelle deux objets sont proches s'ils se situent entre deux autres objets proches l'un de l'autre.

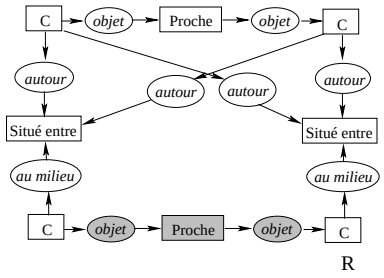


FIG. 5.37 – Une règle établie à partir du vocabulaire de géolocalisation.

Lorsqu'une annotation décrivant une offre touristique est ajoutée à $tm_{\mathcal{A}}$ par le service d'annotation, elle est enrichie par les règles de géolocalisation et validée par rapport aux contraintes puis intégrée parmi les connaissances d'ITM. La figure 5.38 illustre l'état d'une base ITM après intégration de l'annotation enrichie et validée. Les sommets noirs représentent l'annotation extraite par le service d'annotation, les sommets gris le contenu inféré et les sommets blancs le contenu initial. La proximité entre l'hôtel `Majestik` et le théâtre `Le Splendide` a été déduite à partir de la règle R de la figure 5.37 et du fait que l'hotel et le théâtre sont situés entre deux monuments proches l'un de l'autre.

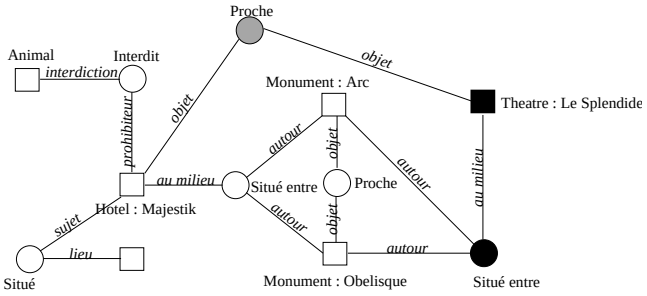


FIG. 5.38 – Espace des annotations de géolocalisation après enrichissement et validation d'une nouvelle annotation.

La phase de diffusion est réalisée à partir d'un site web dont l'ergonomie tient compte des propriétés sémantiques des ressources enrichies que l'utilisateur consulte. En plus d'un service de recherche et de navigation parmi les offres, un mécanisme automatique de suggestions, qui s'appuie sur une base $B'_{\mathcal{SG}}$ de raisonnement $\mathcal{SG}$, propose à l'utilisateur de nouvelles offres susceptibles de l'intéresser. Ce mécanisme est conçu pour couvrir à la fois les besoins de l'utilisateur qui cherche la ressource touristique la plus adaptée à ses préférences et ceux de l'ONT qui doit promouvoir ces ressources.

De manière générale, le mécanisme fournit un ensemble de *requêtes de suggestion* sous la forme de SG obtenues (1) à partir des *ressources choisies* par l'utilisateur et (2) conformément aux *règles promotionnelles* définies par l'ONT. Le mécanisme de suggestion prend en argument une liste de type de ressources touristiques et n'utilise que le service d'interrogation de la base $B'_{\mathcal{SG}}$ (les ensembles de règles et de contraintes de cette base sont donc vides). Un élément de cette liste pourrait être un SG représentant "un hotel trois étoiles se situant entre un théâtre et un restaurant".

Le système utilise ensuite des *règles promotionnelles* pour construire les suggestions. Une règle promotionnelle est un SG bicoloré dont l'un des sous-graphes colorés exprime une *condition* sur la ressource choisie et l'autre est la *suggestion* qui sera complétée par des connaissances de la base d'assertions. Un exemple de règle promotionnelle serait *si un utilisateur a choisi un hôtel parce qu'il est proche d'un théâtre, alors suggérer un restaurant proche de ce théâtre*. Le théâtre dans chaque partie de la règle désigne une même entité et correspond donc à un sommet frontière.

Pour chaque SG représentant une ressource touristique choisie par l'utilisateur, on réalise un ajout contrôlé ; la ressource étant acceptée sans qu'aucune inférence n'ait lieu puisque les ensembles de règles et de contrainte sont vides. Dans la suite, G_T est le sous-graphe induit par les sommets ajoutés dans la base d'assertions de $B'_{\mathcal{SG}}$ et on s'intéresse aux projections π de la partie condition H_P de chaque règle promotionnelle $\mathcal{R}_P$ établies par le service d'interrogation sur au moins un sommet de G_T dans cette base. Le mécanisme de suggestion produit ensuite une *requête de suggestion* qui prend la forme

d'un graphe G_S identique à la règle R_P à ceci près qu'il n'est pas bicoloré et que la partie condition C_P a été spécialisée en le sous-graphe image par π dans la base. La requête de suggestion G_S est ensuite envoyée à un moteur de recherche qui cherche parmi toutes les ressources touristiques celles qui répondent à cette requête. Dans notre exemple, l'entité *Le Splendide* correspondant à un théâtre sert à la définition de la requête de suggestion *trouver un restaurant proche du théâtre Le Splendide"*. Cette dernière est utilisée par le moteur de recherche pour rechercher les ressources correspondant à cette requête qu'il présente à l'utilisateur en tant que suggestions.

5.4 Conclusion

Ce chapitre était consacré au principal résultat de la thèse : la solution qui a été conçue pour doter ITM de capacités d'inférence et de contrôle. Cette solution exploite les transformations entre le langage des TM et la famille SG présentées au chapitre précédent afin d'utiliser les possibilités de représentation et les mécanismes de raisonnement dont dispose la famille $\mathcal{SG}$.

Le chapitre suivant présente la manière dont l'application a été implémentée dans ITM.

Chapitre 6

Implémentation

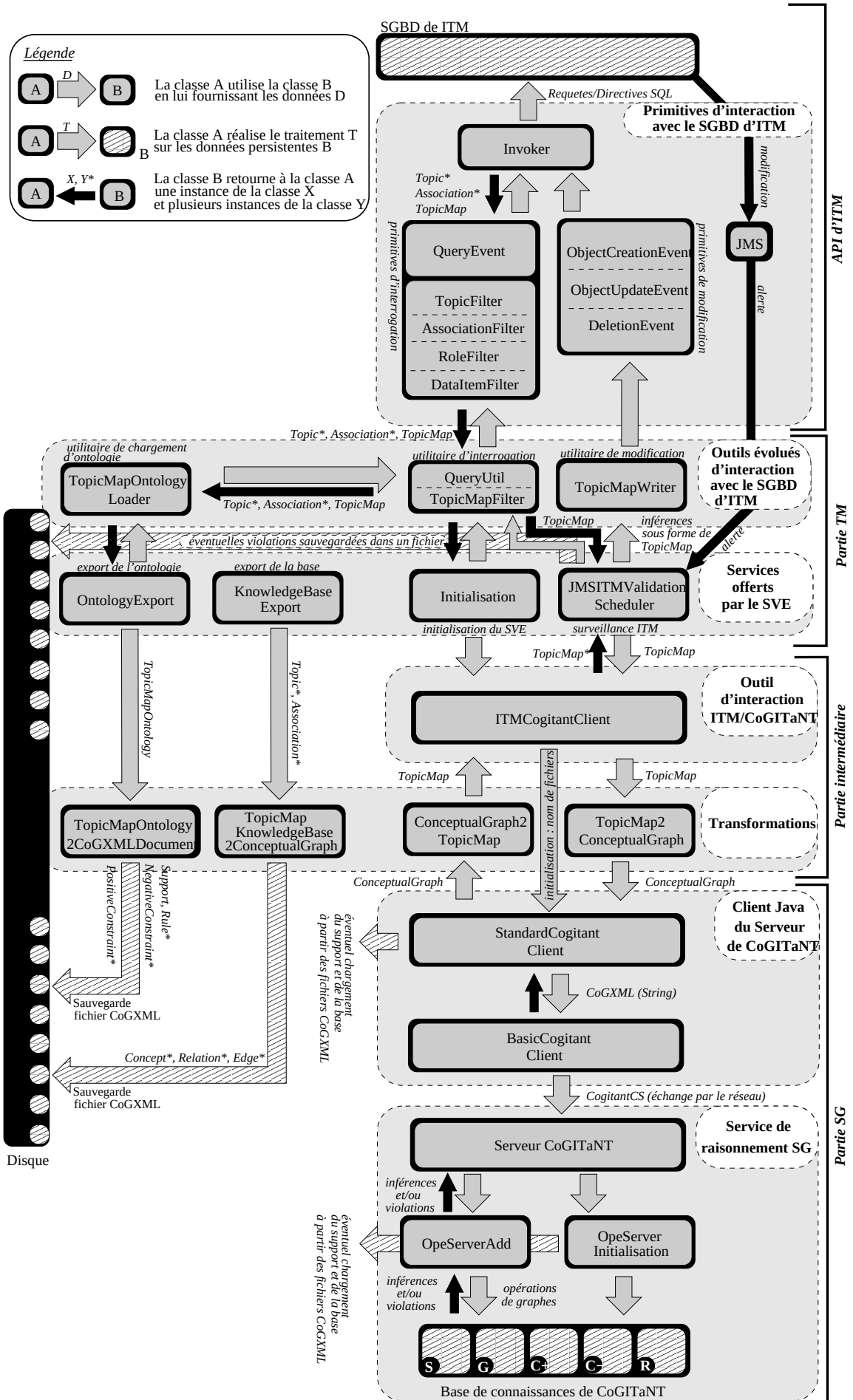
Ce chapitre présente la façon dont le service de validation et d'enrichissement (*SVE*) présenté au chapitre précédent a été implémenté dans ITM. La première section présente l'architecture générale de l'application, la seconde l'API qui permet au *SVE* d'interagir avec ITM et la troisième l'API CoGITaNT qui a été utilisée pour implémenter le service de raisonnement $\mathcal{SG}$. La conception de ce service - et en particulier celle de l'opération d'ajout contrôlé - est décrite en quatrième section. Enfin la cinquième et dernière partie présente l'implémentation du *SVE* puis s'achève par quelques expérimentations réalisées sur des données métiers.

6.1 Architecture

Le *SVE* a été implémenté sous la forme d'une application qui se structure en trois parties : une *partie TM* qui a été développée à partir de l'API d'ITM et ne manipule que des TM, une *partie* $\mathcal{SG}$ s'appuyant sur la famille $\mathcal{SG}$ et une *partie intermédiaire* qui orchestre les échanges entre les deux parties précédentes en exploitant les transformations établies entre le langage des TM et la famille $\mathcal{SG}$. La figure 6.1 présente l'architecture générale du SVE d'ITM.

6.1.1 L'API d'ITM

L'API d'ITM met à disposition deux catégories de *primitives d'interaction* avec le SGBD où sont entreposées les TM : (1) trois *primitives de modification* qui permettent respectivement de créer, modifier ou supprimer un élément dans une TM ; (2) des *primitives d'interrogation* qui permettent de décrire le *profil* d'un topic, d'une association, d'un rôle ou d'une occurrence à chercher dans la base.


Légende
A D B La classe A utilise la classe B en lui fournissant les données D
A T B La classe A réalise le traitement T sur les données persistentes B
A X, Y* B La classe B retourne à la classe A une instance de la classe X et plusieurs instances de la classe Y
SGBD de ITM
Requetes/Directives SQL
Primitives d'interaction avec le SGBD d'ITM
Invoker
Topic*
Association*
TopicMap
modification
JMS
alerte
primitives d'interrogation
QueryEvent
TopicFilter
AssociationFilter
RoleFilter
DataItemFilter
primitives de modification
ObjectCreationEvent
ObjectUpdateEvent
DeletionEvent
API d'ITM
Topic*, Association*, TopicMap
utilitaire de chargement d'ontologie
TopicMapOntology Loader
utilitaire d'interrogation
QueryUtil
TopicMapFilter
utilitaire de modification
TopicMapWriter
Outils évolués d'interaction avec le SGBD d'ITM
Topic*, Association*, TopicMap
TopicMap
inférences sous forme de TopicMap
éventuelles violations sauvegardées dans un fichier
alerte
export de l'ontologie
export de la base
OntologyExport
KnowledgeBase Export
Initialisation
JMSITMValidation Scheduler
Services offerts par le SVE
Partie TM
initialisation du SVE
surveillance ITM
TopicMap*
TopicMap
Outil d'interaction ITM/CoGITaNT
ITMCogitantClient
TopicMapOntology
Topic*, Association*
TopicMap
TopicMap
initialisation : nom de fichiers
TopicMapOntology 2CoGXMLDocumen
TopicMap KnowledgeBase 2ConceptualGraph
ConceptualGraph2 TopicMap
TopicMap2 ConceptualGraph
Transformations
Partie intermédiaire
ConceptualGraph
ConceptualGraph
Client Java du Serveur de CoGITaNT
StandardCogitant Client
CoGXML (String)
BasicCogitant Client
PositiveConstraint*
NegativeConstraint*
Support, Rule*
Sauvegarde fichier CoGXML
éventuel chargement du support et de la base à partir des fichiers CoGXML
Concept*, Relation*, Edge*
Sauvegarde fichier CoGXML
CogitantCS (échange par le réseau)
Service de raisonnement SG
Serveur CoGITaNT
inférences et/ou violations
OpeServerAdd
OpeServer Initialisation
inférences et/ou violations
opérations de graphes
S G C+ C- R
Base de connaissances de CoGITaNT
Partie SG
Disque

6.1.2 La partie TM du *SVE*

Cette partie présente deux niveaux logiciels : les *outils évolués d'interaction avec le SGBD d'ITM* qui sont définis à partir des *primitives d'interaction* d'ITM et *les services offerts par le SVE.*

Les outils évolués d'interaction avec le SGBD d'ITM

Trois outils permettent d'interroger ou modifier le contenu de la base : la classe `TopicMapOntologyLoader` charge le niveau modèle d'ITM dans une structure de données `TopicMapOntology` prévue à cet effet ; les classes `QueryUtil` et `TopicMapFilter` fournissent des méthodes d'interrogation prêtes à l'emploi ; et la classe `TopicMapWriter` réalise l'ajout spécialisant d'une TM dans la base d'annotations.

Les trois opérations essentiellement utilisées par le *SVE* présenté au chapitre précédent sont : l'ajout spécialisant, la suppression généralisante et la disjonction de TM. L'API d'ITM fournit des opérations élémentaires d'ajout et de suppression d'un topic, d'une association, d'un nom ou d'une occurrence dans une TM. Ces opérations ont été utilisées pour définir une opération d'ajout de TM qui n'est cependant pas encore spécialisante. Le caractère spécialisant consiste à modifier le type d'un topic de la TM par celui de son homologue dans la TM passée en argument si ce dernier est plus spécifique que le premier. Il reste encore à implémenter la suppression généralisante et la disjonction d'une TM. Pour le moment, les opérations élémentaires de suppression et de modification sont utilisées à la place de ces deux opérations.

Les services offerts par le SVE

Ce niveau présente quatre services :

- Les classes `OntologyExport` et `KnowledgeBaseExport` sauvegardent dans des fichiers sur le disque le résultat de l'application des transformations TM/$\mathcal{SG}$ sur le niveau modèle et la base d'annotations afin de les conserver pour un usage ultérieur.
- La classe `Initialisation` permet de charger l'ontologie et éventuellement la base d'annotations d'ITM dont le contenu a été au préalable traduit, exporté dans le formalisme de la famille $\mathcal{SG}$ et localement entreposé dans des fichiers sur le disque (après un appel à `OntologyExport` et `KnowledgeBaseExport`). Le *SVE* décrit au chapitre 5 prévoit, à l'initialisation, de morceler la TM associée à l'espace des annotations, si ce dernier n'est pas vide. Le processus de décomposition de cet espace en annotations n'étant pas encore tout à fait achevé, la classe `KnowledgeBaseExport` est pour le moment utilisée pour traduire l'espace des annotations en un seul SG sauvegardé dans un fichier sur le disque qui sera ensuite passé en argument de l'ajout contrôlé pour initialiser la base du service de raisonnement $\mathcal{SG}$.

– Une fois le *SVE* initialisé, `JMSITMValidationScheduler` permet de lancer le programme de surveillance d'ITM. Ce dernier reste à l'écoute des alertes envoyées[1] par ITM au moment d'une modification de la base. Chaque alerte reçue est analysée pour reconstituer les annotations créées ou modifiées afin d'évaluer si leur enrichissement et validation est nécessaire. Le résultat de l'enrichissement et de la validation est ensuite sauvegardé dans la base et sur le disque. L'application actuelle traduit le contenu inféré en TM et l'ajoute dans l'espace $E_{\mathcal{A}}$ des annotations. Les éventuelles violations de contraintes sont pour le moment sauvegardées dans un fichier sur le disque. La possibilité de les représenter dans un espace d'ITM qui leur serait réservé est à l'étude.

6.1.3 La partie intermédiaire du *SVE*

Pour déclencher l'enrichissement et la validation d'une annotation, le *SVE* s'appuie sur la classe `ITMCogitantClient` de la partie intermédiaire de l'architecture. Cette dernière applique les transformations entre le formalisme des TM et celui de la famille $\mathcal{SG}$ pour obtenir une annotation SG. En amont de cette partie, les structures de données qui circulent sont essentiellement celles définies dans l'API d'ITM, à savoir des objets `TopicMap`, `Topic` et `Association`. Au delà de cette partie, les structures de données utilisées par le *SVE* correspondent aux composants de la famille $\mathcal{SG}$, à savoir des objets `Support`, `ConceptualGraph`, `Rule`, `NegativeConstraint`, `PositiveConstraint`.

6.1.4 La partie $\mathcal{SG}$ du *SVE*

Cette partie est composée de deux niveaux logiciels : Le *client Java du serveur CoGITaNT* qui fonctionne sur la même machine que les composants logiciels des parties précédentes et le *service de raisonnement* $\mathcal{SG}$ qui peut être hébergé sur une machine distante.

Le client Java du serveur CoGITaNT

Le *client Java du serveur CoGITaNT* est sollicité par la classe `ITMCogitantClient` de la partie précédente pour réaliser l'ajout contrôlé du *SG* obtenu par traduction de l'annotation TM. Ce dernier est traduit en un document XML puis envoyé par liaison TCP/IP au *service de raisonnement* $\mathcal{SG}$ qui retourne sous le même format les éventuelles inférences et violations au client Java. Ces informations circulent de niveau en niveau, et après avoir été traduites en TM parviennent jusqu'au programme de surveillance d'ITM qui les sauvegarde dans un fichier sur le disque et dans le SGBD d'ITM.

[1] Les alertes sont gérées par une entité de la machine virtuelle `Java` appelée le `Java Message Service` (`JMS`).

Le service de raisonnement $\mathcal{SG}$

Le service de raisonnement $\mathcal{SG}$ permet d'effectuer deux opérations sur la base (en dehors de son initialisation) : l'ajout contrôlé d'une annotation SG et l'interrogation de la base (cette dernière étant fonctionnelle mais pas directement utilisée par le *SVE*). Il est prévu d'implémenter prochainement les opérations d'ajout contrôlé d'une règle et d'une contrainte, le retrait d'une contrainte et l'identification des sous-graphes de la base *réparés* par l'ajout.

La fonction d'identification i n'est pas encore implémentée dans la base $B_{\mathcal{SG}} = (S, G, R, C^+, C^-)$. Les sommets génériques de G ne peuvent donc pas être identifiés et tout ajout contrôlé d'une annotation SG qui possède n sommets concepts génériques produit donc exactement n nouveaux sommets concepts génériques sans les fusionner à ceux déjà présents dans G. Ceci peut conduire dans certains cas peu fréquents à une désynchronisation des deux bases d'annotations. Par exemple, on considère une annotation *valide* qui présente un topic non identifié[2] t ayant été traduit en un sommet concept générique c dans G. On créé une nouvelle annotation en utilisant le topic t (on lui rajoute une occurrence par exemple). Après validation et enrichissement de la nouvelle annotation, le sommet correspondant dans G à la traduction de t n'est pas le même que c : les deux bases sont désynchronisées.

6.1.5 Réparation et amélioration

La réparation d'une annotation est réalisée par suppression des constituants élémentaires à l'origine de la violation ; chaque annotation *invalide* concernée par une suppression étant à nouveau soumise à l'ajout contrôlé. L'amélioration d'une annotation *incomplète* pourrait être réalisée par ajout de TM ; cependant comme le service de raisonnement $\mathcal{SG}$ est encore dans l'incapacité d'identifier les parties de la base réparées par l'ajout, il n'est pas encore possible de savoir si l'ajout conduit ou non à une amélioration.

6.2 L'API d'ITM

ITM fournit une API écrite en Java qui permet d'interroger et de modifier la base de données où sont entreposées les TM. Toute opération sur la base est réalisée à partir d'un objet de classe `Invoker` qui se connecte à la base et interprète les instructions transmises en traitements sur les données de la base. Ces instructions sont scindées en deux classes : les instructions élémentaires d'interrogation, d'ajout ou de suppression d'éléments et celles de plus haut niveau qui ont été développées à partir des précédentes. L'utilisation de l'invoker peut être surveillée par un mécanisme appelé *module d'alerte* de telle sorte

[2]Un topic non identifié ne possède pas de PSI (dans ITM les topics sont identifiés par leur PSI et non par leur nom). Ce cas est rare car par convention tout topic de ITM doit porter un PSI.

qu'à chaque instruction reçue par l'invoker le programme requérant cette surveillance soit prévenu de la nature de l'instruction (interrogation, ajout ou suppression) et des données qu'elle cible dans la base.

La section suivante présente la structure de donnée qui permet de modéliser une TM en Java suivie des instructions élémentaires d'ajout, de suppression et d'interrogation utilisées pour développer le service de validation et d'enrichissement d'ITM. La dernière section présente les principales fonctionnalités du module d'alertes.

6.2.1 La classe TopicMap

Les TM d'ITM sont entreposées dans un SGBD ; une table du SGBD étant réservée à chaque élément : association, topic, rôle ou occurrence. Généralement les données de la base ne sont pas directement accessibles. Toute application développée à partir de l'API d'ITM utilise la classe `Invoker` pour réaliser des requêtes dans la base. Le résultat d'une requête est une structure de donnée Java qui permet de représenter ces données brutes sous une forme plus proche du standard : la classe `TopicMap`.

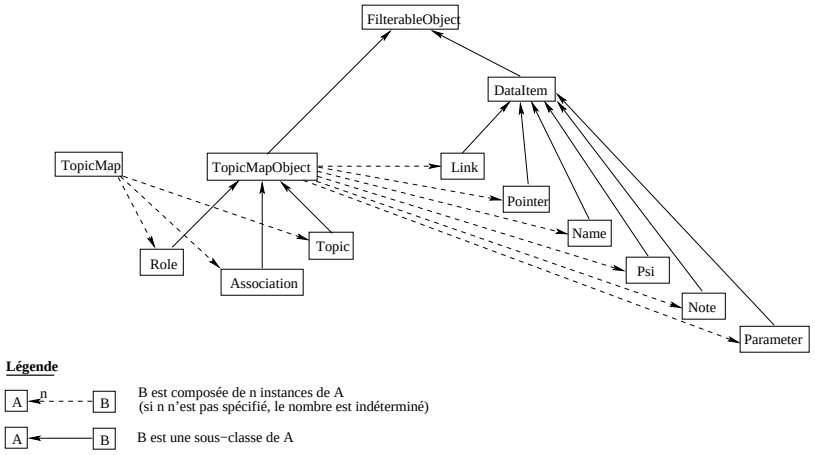


FIG. 6.2 – Classes modélisant le formalisme Topic Map.

La figure 6.2 présente la hiérarchie des classes modélisant le formalisme Topic Map et les relations de composition entre ces classes. Un objet de classe `TopicMap` contient une liste de topics (classe `Topic`) et une autre d'associations (classe `Association`). Chaque objet `Topic` ou `Association` dispose d'un type et d'une liste de `Role`. Il est de plus sous-classe de `TopicMapObject` qui est détenteur d'une liste de `DataItem`. La classe `DataItem` modélise la notion d'occurrence. Chacune de ses sous-classes `Link`, `Name`, `Note`, `Parameter`, `Pointer` et `Psi` modélise une occurrence d'un type physique particulier ; le

nom d'un topic étant représenté par l'API comme une occurrence de type `Name` ; et non pas comme un attribut spécifique à la classe `Topic`. Chacune de ces classes est instanciée en spécifiant le type logique de l'occurrence. Une fois l'occurrence instanciée sa valeur peut être fixée.

La classe `TopicMap` est surtout intéressante pour représenter le résultat d'une requête d'interrogation, c'est à dire exclusivement des connaissances de la base. Elle n'est pas destinée à être instanciée par une application autre que l' `Invoker`. Cependant, dans certains cas exceptionnels, il se peut qu'elle soit utilisée pour représenter temporairement des connaissances qui ne se trouvent pas dans la base.

6.2.2 Invoker

Pour instancier un objet de classe `Invoker` en tant que client d'ITM, il faut fournir des informations confidentielles de connexion à la méthode `connect`. Ces informations sont essentiellement[3] le nom d'utilisateur, le mot de passe et la société cliente du service ITM.

Chaque instruction d'ajout, de suppression ou d'interrogation est passée à l'objet `Invoker` par l'intermédiaire de la méthode `perform` qui prend en paramètre l'instance d'une classe implémentant l'interface `MondecaEvent`. Il existe une trentaine de classes de ce type. Celles réservées aux opérations les plus élémentaires sont présentées dans la suite.

Opération d'ajout et de modification

L'ajout d'un topic t ou d'une association a d'un certain type est réalisé à partir d'une instance de `TopicCreationEvent` ou `AssociationCreationEvent`. Après avoir fourni l'instruction à la méthode `perform`, cette dernière retourne l'identifiant de l'élément créé ou lève une exception si l'accès à la base est trop restreint pour réaliser l'opération.

S'il est nécessaire de créer des occurrences sur le topic ou l'association, on peut directement le faire en instanciant un objet `DataItemCreationEvent` auquel on a spécifié la valeur et les types (logique et physique) de l'occurrence ; cet objet étant ajouté à la liste des occurrences à créer de l'instruction `TopicCreationEvent` ou `AssociationCreationEvent`.

Il en va de même pour les rôles : la création d'un rôle d'un certain type entre l'association a et un topic x de la base peut être réalisée en instanciant un objet `RoleCreationEvent` qui sera placé dans la liste des rôles à créer de `AssociationCreationEvent`.

[3] D'autres informations plus techniques qui identifient le serveur et le type de connexion à établir peuvent être nécessaires.

Opération de suppression

L'opération de suppression d'un topic, d'une association, ou d'un rôle est réalisée à partir d'une instance de `DeletionEvent`. Il suffit de fournir l'identifiant de l'élément à supprimer à l'instanciation de l'objet puis de passer cet objet à la méthode `perform()` de l'invoker. Le mécanisme supprime ensuite l'élément indiqué et tous les autres éléments dont l'existence dépendait de la sienne. Par exemple, lorsqu'une association est supprimée toutes ses occurrences et ses rôles le sont aussi. Il en va de même pour les topics.

Opération de modification

Pour ajouter ou supprimer des occurrences ou des métadonnées à des topics, des associations ou des rôles déjà présents dans la base, il est nécessaire d'utiliser l'opération de modification `ObjectUpdateEvent`. Cette opération est instanciée avec l'identifiant de l'élément à modifier. On lui fournit la liste des occurrences ou métadonnées à ajouter ou supprimer.

Opération d'interrogation

L'instruction `QueryEvent` permet d'interroger la base de connaissances d'ITM. Dans ce cas, la méthode `perform` de l'invoker retourne un objet `TopicMap`. Une requête sur la base ITM se construit en combinant divers *filtres* qui s'organisent en hiérarchie et décrivent le *profil* des objets cherchés. Le résultat de cette requête correspond à la connaissance de la base qui n'a pas été éliminée par les filtres. Il existe deux hiérarchies possibles selon la cible de la requête.

Lorsque la requête porte sur les associations (voir la fonction i_a du chapitre 5), le sommet de cette hiérarchie est le filtre `AssociationFilter`. On fournit à ce filtre, des objets `RoleFilter` et on ajoute à ces derniers des objets `TopicFilter` ; chacun de ces objets ciblant respectivement les rôles et les topics.

Lorsque la requête porte sur les topics (voir la fonction i_t du chapitre 5), le sommet de la hiérarchie des filtres est le filtre `TopicFilter` qui peut contenir des filtres `RoleFilter`, eux-mêmes parents d' `AssociationFilter`. Les filtres d'associations regroupent à nouveau des filtres `RoleFilter` et la hiérarchie se termine par des `TopicFilter`.

Dans les deux cas, les filtres d'association et de topic peuvent être parents de `DataItemFilter`, s'il est nécessaire de focaliser la recherche sur les éléments portant des occurrences bien spécifiques. Ils peuvent aussi rassembler des objets `TopicProperties` ou `AssociationProperties` qui permettent d'inclure ou d'exclure des occurrences dans le résultat de la requête.

6.2.3 Alertes

Le module d'alertes d'ITM permet à des applications externes de recevoir de manière asynchrone des messages provenant d'ITM. La clé de voûte de ce système est le gestionnaire de messages Java appelé JMS (Java Message Service). Pour qu'une application reçoive des alertes à chaque modification de la base, il faut paramétrer l'invoker de telle sorte qu'il envoie à chaque création, modification ou suppression dans la base un message au JMS. Il faut d'autre part étendre la classe `JMSMessageListener` en une classe qui surcharge la méthode `onMessage()` à laquelle le JMS va transmettre le message d'alerte reçu de la part de l' `Invoker` sous la forme d'un objet de classe `AlertResultMessage`. Le code de l'application à exécuter à la reception de l'alerte doit être appelé par cette méthode.

L'exécution d'une application qui s'insère dans cette architecture logicielle est réalisée parallèlement à celle des traitements de ITM. Le système global ainsi constitué par ITM et la dite application peuvent conduire à des performances plus intéressantes qu'un système où l'exécution de l'un se déroulerait séquentiellement à celle de l'autre.

6.3 CoGITaNT

CoGITaNT (Conceptual Graphs Integrated Tools allowing Nested Typed graphs) est une bibliothèque de classes C++ permettant le développement de logiciels basés sur les graphes conceptuels. En plus des opérations habituelles permettant la manipulation de graphes conceptuels en mémoire (ajout de sommet, suppression de sommet, changement d'étiquette, etc.), la bibliothèque offre un mécanisme de lecture et d'écriture de fichiers au format BCGCT (format natif de représentations des connaissances), au format CoGXML (format xml de représentations des connaissances), des opérations permettant de vérifier qu'un graphe conceptuel est correctement construit (vérification des signatures et de la relation de conformité), ainsi que l'opération de projection. La bibliothèque prend en compte les extensions du modèle proposées dans différents travaux telles que les règles de graphes conceptuels (et les opérations associées).

Une architecture client-serveur a été développée en utilisant cet ensemble de classes. La communication entre le client et le serveur est basée sur le protocole TCP/IP. Toutes les opérations coûteuses (en espace mémoire ou en ressource processeur) se font sur la machine hébergeant le serveur, et il n'est pas nécessaire que la machine faisant fonctionner le client soit très puissante.

Toutes les sources des différents modules formant CoGITaNT peuvent être téléchargées gratuitement à partir du site Web [Gen97], ainsi que des versions précompilées[4] pour certaines plateformes.

[4]Ces sources ont été testées sous différents systèmes avec différents compilateurs.

Les sections suivantes résument les objets et fonctionnalités de la bibliothèque qui ont été utilisées. Il est à noter que CoGITaNT offre des outils et des objets pour représenter le formalisme des graphes conceptuels emboîtés typés. Ce formalisme n'ayant pas été utilisé dans le cadre de ce travail, la partie de CoGITaNT implémentant ces graphes (et les opérations qui leur sont associées) n'est donc pas présentée. La section suivante décrit les classes définies pour représenter en C++ le formalisme $\mathcal{SG}$. Quand aux sections qui suivent, elles présentent les classes implémentant les opérations, le serveur et les formats d'échanges ou de stockage qu'offre CoGITaNT.

6.3.1 Représentation du formalisme $\mathcal{SG}$

CoGITaNT permet de représenter des graphes conceptuels, des règles ou des contraintes au sein d'*environnements* définis à partir d'un support ; le nombre d'environnements n'étant limité que par les ressources de la machine.

Le support

La classe `Support` offre une représentation d'un support (voir chapitre 2) formé d'un ensemble de types de concepts, un ensemble de types de relations et un ensemble de marqueurs individuels. Les ensembles de types sont partiellement ordonnés par une relation "sorte de". Enfin, la relation de conformité, τ, associe à tout marqueur individuel un type de concept et la relation σ associe à tout type de relation sa signature, c'est à dire le type de concept maximal de chacun de ses arguments.

Les ensembles de types de concepts, types de relations, et marqueurs individuels sont accessibles au travers des méthodes `conceptTypes()`, `relationTypes()` et `individuals()`. Ces ensembles sont définis sous la forme de `Set` de `ConceptType`, `RelationType` et `Individual`.

Graphe conceptuel

Un graphe conceptuel (`Graph`) est représenté sous la forme d'un ensemble de sommets (`GraphObject`) et d'un ensemble d'arêtes (`Edge`). Les classes `Concept` et `Relation` représentent les sommets concepts et relations.

Pour créer un nouveau sommet concept individuel ou générique il faut respectivement appeler les méthodes `newIndividualConcept()` et `newGenericConcept()` auxquelles l'identificateur du type doit être fourni, la première prenant en plus l'identificateur du marqueur en paramètre.

La méthode `link()` permet de lier un sommet relation à un sommet concept. Cette méthode doit recevoir comme paramètres l'identificateur d'un sommet relation, une étiquette d'arête et l'identificateur d'un sommet concept.

La suppression de sommets d'un graphe se fait en appelant la méthode `deleteObject()` qui prend comme paramètre l'identificateur du sommet à détruire.

Les liens de coréférence ne sont pas explicitement représentés entre sommets concepts, mais à travers des classes de coréférence instances de `CoreferenceClass`.

Les règles

Une règle est composée de deux graphes conceptuels ainsi que d'un ensemble de points de connexions. Dans CoGITaNT, la structure de données d'une règle s'inspire de la définition du modèle. La classe `Rule` possède deux attributs qui sont des graphes (l'hypothèse et la conclusion) et d'un ensemble de couples qui sont les points d'attache. La méthode `hypothesis()` retourne le graphe hypothèse, et `conclusion()` le graphe conclusion. La méthode `connectionPoints()` fournit l'ensemble des couples représentant les point de connexions : le premier élément d'un couple est l'identificateur du sommet dans le graphe hypothèse, et le second élément est l'identificateur du sommet dans le graphe conclusion.

Les contraintes

Les contraintes sont représentées sous la forme de graphes bicolorés. La classe `Constraint` est donc une sous classe de `ColoredGraph`, qui est une sous-classe de `Graph`. Une contrainte peut donc être considérée comme un graphe, et les opérations classiques (projections, entrées sorties, etc.) des graphes peuvent être utilisées sur les instances de `Constraint`. La seule différence entre un graphe contrainte et un graphe est la présence d'une propriété `GRAPH_NATURE`, définie à `PositiveConstraint` ou `NegativeConstraint` pour les graphes-contraintes, et la propriété `COLOR` pour les sommets du graphe.

Représentation d'une base de connaissances $\mathcal{SG}$

La classe `Environment` permet la gestion d'un support, de graphes et de règles définies sur ce support. Elle offre un accès facilité aux opérations faisant intervenir supports, graphes et règles, telles que les opérations d'entrées sorties et les opérations définies dans le modèle des graphes conceptuels décrites dans la section suivante.

6.3.2 Représentation des opérations du formalisme $\mathcal{SG}$

Les classes implémentant une opération doivent étendre la classe `Operation`. Un vingtaine d'opérations sont ainsi programmées dans l'API CoGITaNT. Les données de chaque opération sont fournies en utilisant les méthodes dont le nom est composé du préfixe `setParam-`. Le déclenchement est réalisé à partir de la méthode `run()` et le résultat est obtenu par la méthode d'accès dont le nom présente le préfixe `getResult-` .

Opération de projection

La recherche de projections d'un graphe dans un autre se fait à l'aide de la méthode `projections()`, à qui on doit fournir le graphe source et le graphe cible de la projection. Le troisième paramètre de cette méthode, un `ResultOpeProjection` est modifié par l'exécution, et contient après l'appel le résultat de l'opération.

Comme toutes les autres méthodes de `Environment` qui donnent accès à des opérations, `projections()` n'est qu'un raccourci qui utilise des sous-classes de `Operation`. En réalité, c'est la classe `OpeProjection` qui réalise le calcul des projections en s'appuyant sur un algorithme de backtrack. Ce dernier calcule dans un premier temps les listes d'images possibles de chaque sommet du graphe source dans les sommets du graphe cible, puis filtre ces listes.

La suite de cette section présente le fonctionnement standard de l'opération de projection. Cette opération est modulable, c'est à dire qu'elle fait appel à d'autres opérations spécialisées dans une tâche qu'il suffit de redéfinir et de remplacer pour changer le comportement de la projection. Le second paragraphe énumère les opérations spécialisées qu'utilise l'algorithme de projection.

Projection standard

En plus de contenir les projections trouvées, l'objet `ResultOpeProjection` permet de configurer la recherche. En effet, selon les besoins, on peut être intéressé par différentes recherches :

- Si c'est uniquement l'existence d'une projection entre les deux graphes qui interesse l'utilisateur, il est inutile que l'opération mémorise au fur et à mesure de la recherche les projections trouvées. Dans ce cas, il est conseillé d'appeler les méthodes `memoProjections()` et `maxSize()` respectivement avec les paramètres false et l'entier 1. Elles permettent de limiter la recherche à une seule projection (la première trouvée) sans la mémoriser.
- Lorsque l'utilisateur cherche à savoir le nombre de projections, on conserve la configuration précédente à ceci près que `maxSize()` n'est pas appelée. La méthode `size()` permet de connaître le nombre de projections trouvées.
- Lors d'un usage standard, l'utilisateur cherche l'ensemble des projections d'un graphe dans un autre, et souhaite manipuler la projection elle-même, c'est à dire, l'ensemble des couples constitués du sommet projeté et de son image. Dans ce cas, aucune méthode particulière de `ResultOpeProjection` ne doit être appelée, et après l'appel à `projections()`, la méthode `projections()` retourne l'ensemble des projections trouvées.

Modularité de l'opération de projection

L'opération définie en `OpeProjection` n'est en réalité qu'un algorithme orchestrant de manière générale la recherche des projections en faisant appel à d'autres opérations, spécialisées dans une tâche. Cette architecture modulaire permet à l'opération d'être facilement modifiée. Il est possible d'une part d'écrire une sous-classe de `OpeProjection` s'il est nécessaire de redéfinir l'algorithme de backtrack, ou d'autre part d'étendre une des classes définissant les opérations spécialisées s'il est suffisant de ne modifier qu'une partie précise du comportement de la recherche pour trouver les projections souhaitées. On dénombre sept opérations de ce type présentées dans la suite :

- `OpeProjPrecalcImages` permet de déterminer le moment auquel la liste d'images possibles d'un sommet projeté doit être calculé (avant ou pendant l'exécution du backtrack).
- Si le calcul doit avoir lieu avant le lancement du backtrack, l'opération `OpeProjLIPInit` s'en charge.
- Le calcul d'une liste d'images possibles dépend de la "compatibilité" de sommets pour savoir si un sommet d'une étiquette donnée peut être projeté sur un autre sommet d'une étiquette donnée. Ce calcul est réalisé par `OpeGraphObjectCompatibility`.
- `OpeProjBacktrackChoice` est utilisé pour choisir au cours du mécanisme de backtrack le prochain sommet dont la liste d'images possibles doit être filtrée.
- `OpeProjLIPUpdate` est l'opération définie pour filtrer une liste d'images possibles.
- `OpeProjAcceptableCouple` est définie pour accepter ou refuser un couple dans la projection.
- `OpeProjAcceptableLIPs` détermine si les listes d'images possibles à un moment donné de la recherche respectent certains critères.

Opération de vérification des contraintes

La satisfaction d'une contrainte par un graphe peut être vérifiée par un appel à la méthode `constraintSatisfaction()` prenant en paramètres le graphe à vérifier et la contrainte et retournant true ou false selon que le graphe satisfait ou pas la contrainte. Si on désire connaître les projections identifiant un sous-graphe du graphe à vérifier et violant la contrainte positive ou négative, il faut fournir à la méthode `constraintSatisfaction()` un troisième paramètre, optionnel, qui est un pointeur sur un `ResultOpeProjection`.

Opérations réservées aux règles

Trois opérations sont accessibles à travers des sous-classes de `Operation` :

- `OpeRuleApplications` détermine les applications possibles d'une règle sur un graphe et retourne les projections de l'hypothèse de la règle sur le graphe ;

– `OpeRuleApply` réalise l'application d'une règle sur un graphe selon une projection.
– `OpeRulesClosure` se charge de la saturation ou fermeture d'un graphe par rapport à un ensemble de règles. Cette opération consiste à appliquer les règles de l'ensemble considéré, jusqu'à ce que toute application supplémentaire pour toute règle de cet ensemble ne fasse qu'ajouter de la redondance.

6.3.3 Les formats d'échange et de stockage

`CoGITaNT` met à disposition plusieurs formats d'échange et de stockage : le `BCGCT`, le `CoGXML` et le `CogitantCS`.

Le format `BCGCT` (Base de Connaissances Graphes Conceptuels Textuelle) est un format plus ancien que le `CoGXML` qui permet de représenter les différents composants de la famille $\mathcal{SG}$. Il n'a pas été utilisé pour ce projet.

Le format `CoGXML` permet de représenter les constituants de la famille $\mathcal{SG}$ sous la forme de documents XML. La DTD[5] permet la représentation de supports, de graphes, de règles et de contraintes sous une forme qui peut facilement être interprétée par des humains et par des programmes.

Plutôt que d'éditer des graphes ou supports en `CoGXML` à partir d'un éditeur de texte, il est possible d'utiliser des éditeurs graphiques capables de gérer le format `CoGXML` tel `CoGUI` [Gut06] ou `TooCoM` [Fur04].

Les échanges entre un serveur et un client se font dans un format XML appelé `CogitantCS` défini par une DTD[6]. Chaque document en `CogitantCS` contient des directives qui encapsulent du CoGXML. Ce langage n'est cependant pas utilisé par le serveur et le client conçus pour le *SVE*. Il a été remplacé par un nouveau langage appelé `SVECogitantCS` dont la DTD définit cinq nouvelles directives correspondant aux cinq services offerts par $B_{\mathcal{SG}}$. La figure 6.3 présente la plus grande partie de cette DTD. La prise en charge de ces cinq directives par le serveur est détaillée dans la section suivante.

[5] `http://`cogitant.sourceforge.net/cogxml.dtd
[6] `http://`cogitant.sourceforge.net/cogitant_html/prog_cs.html

```
<!ELEMENT qinitialisation EMPTY>
<!ATTLIST qinitialisation file CDATA #IMPLIED>

<!ELEMENT qsupport EMPTY>
<!ATTLIST qsupport>

<!ELEMENT qknowledgebase EMPTY>
<!ATTLIST qknowledgebase>

<!ELEMENT qadd (#PCDATA)>
<!ATTLIST qadd file CDATA #IMPLIED  embedded CDATA #IMPLIED>

<!ELEMENT qquery (#PCDATA)>
<!ATTLIST qquery>
```

FIG. 6.3 – La partie principale de la DTD du langage d'échange client-serveur `SVECogitantCS`.

6.3.4 La classe du serveur

L'architecture client-serveur de CoGITaNT permet de rendre accessibles les objets et services offerts par la bibliothèque à d'autres applications pouvant être localisées sur des machines distantes. Cette architecture client/serveur permet le développement de clients légers épousant les besoins de l'utilisateur en matière de gestion des connaissances. De tels clients peuvent se connecter au serveur en liaison TCP/IP (ou par le protocole HTTP, cas qui n'est pas détaillé dans ce document), communiquer en formulant les messages en `CogitantCS` (ou ses dérivés tels que le `SVECogitantCS`) et exploiter les services fournis par le serveur. Il n'est pas nécessaire que ces clients utilisent la bibliothèque CoGITaNT, pour représenter les informations reçues du serveur (ils peuvent utiliser leurs propres structures de données).

6.4 Serveur de raisonnement $\mathcal{SG}$

Le serveur de raisonnement $\mathcal{SG}$ développé pour ce projet spécialise la classe du serveur fournie par CoGITaNT, désactive les opérations standards de CoGITaNT et active cinq opérations `OpeServerInitialisation`, `OpeServerSupport`, `OpeServerKnowledgeBase`, `OpeServerQuery` et `OpeServerAdd` chacune d'elles étant respectivement utilisée pour :

(1) initialiser le serveur au démarrage,
(2) sauvegarder en CoGXML le support et (3) la base dans un fichier sur le disque de la machine hébergeant le serveur,
(4) interroger la base d'assertions
(5) réaliser l'ajout contrôlé d'une annotation SG.

Ces opérations sont définies dans 35 fichiers écrits en C++[7] et leur déclenchement est assuré par une classe qui spécialise la classe `Server`. Il peut être pertinent de réserver une machine à l'usage exclusif du serveur car les opérations qu'il réalise exigent parfois des ressources importantes (en temps, en mémoire et en puissance de calcul). Cette classe interprète les directives `SVECogitantCS` pour déterminer l'opération adéquate à exécuter. Lorsqu'un document `CoGXML` encapsulé dans une directive `SVECogitantCS` et envoyé au serveur ne respecte pas les DTD associées aux deux langages le serveur retourne un message d'erreur identifiant la partie du document posant problème et la contrainte violée.

L'opération `OpeServerInitialisation` se déclenche à la réception d'un document CoGXML encapsulé dans la directive SVECogitantCS `qinitialisation`. Ce document contient le support, les règles et les contraintes de la base de connaissances du serveur. L'opération ne retourne rien si elle s'est déroulée avec succès.

L'opération `OpeServerQuery` consiste à rechercher les projections dans la base d'assertions du graphe fourni en argument en utilisant l'opération standard fournie par CoGITaNT : `OpeProjection`. L'opération se déclenche à la réception du graphe H en CoGXML encapsulé dans une directive `qquery`. L'opération retourne l'ensemble des sous-graphes de la base G images de H par projection.

Les opérations `OpeServerSupport` et `OpeServerKnowledgeBase` sont respectivement déclenchées à la reception des directives `qsupport` et `qknowledgebase`. Elles permettent de sauvegarder, dans un fichier sur le disque de la machine hébergeant le serveur, le support et la base (incluant les règles, les contraintes et le graphe d'assertions) pour éviter une éventuelle perte de données.

Enfin, l'opération `OpeServerAdd` consiste à enchaîner les étapes de *contrôle*, d'*intégration*, d'*enrichissement* et de *validation* (décrites au chapitre précédent) en intercalant entre chacune d'elles une étape de vérification des contraintes négatives. Ces étapes sont respectivement implémentées dans les classes `OpeControl`, `OpeTargetedNormalisation`, `OpeTargetedRulesClosure` et `OpeTargetedPositiveConstraintValidation` ; la *vérification des contraintes négatives* étant réalisée par `OpeTargetedNegativeConstraint-Verification`.

L'enrichissement, la validation et la vérification des contraintes négatives sont réalisés de manière incrémentale. Les traitements au cours de ces étapes, utilisent à la place de la projection standard de CoGITaNT l'opération `OpeTargetedProjection` qui concentre la projection sur un ensemble de sommets dans le graphe cible, en l'occurrence les sommets ajoutés lors de l'intégration ou inférés au cours de l'enrichissement. La projection ciblée est décrite dans la section suivante, l'implémentation de l'étape d'enrichissement dans celle qui suit, et l'implémentation de l'étape de validation et de vérification des contraintes négatives dans la dernière section.

[7] 3900 lignes de code

6.4.1 Projection ciblée

Soient deux SG $G = (C_G, R_G, E_G, l_G)$, $H = (C_H, R_H, E_H, l_H)$ et le sous-ensemble C_H^T de l'ensemble des sommets concepts C_H de H, la classe `OpeTargetedProjection` définit une opération de projection d'un graphe *source* G dans le graphe *cible* H telle qu'au moins un sommet de C_H^T soit ciblé par la projection. `OpeTargetedProjection` retourne un ensemble de projections. Le calcul de cet ensemble est réalisé en deux étapes qui consistent respectivement (1) à créer un ensemble $\prod_{partiel}$ de projections partielles et (2) à compléter chacune de ces projections jusqu'à ce qu'elle soit totale, si possible.

Au cours de la première étape, on ajoute dans $\prod_{partiel}$ une projection partielle π avec $\pi(x) = y$ ssi y est plus spécialisé[8] que x, pour chaque paire de sommets concepts x et y appartenant respectivement au graphe *source* et à l'ensemble ciblé C_H^T. Pour vérifier que le sommet y est plus spécialisé que le sommet x, on utilise l'opération `OpeGraphObject-Compatibility`.

Ensuite, on réalise la seconde étape : pour chaque projection partielle π de $\prod_{partiel}$, on fournit π, le graphe G et le graphe H à l'opération standard `OpeProjection` de projection en utilisant les méthodes `setParamProj`, `setParamG` et `setParamH`. `OpeProjection` calcule toutes les projections π' totales de G dans H que l'ont peut obtenir en complétant la projection π. Pour chaque projection partielle π, `OpeProjection` retourne un objet de classe `ResultOpeProjection` contenant l'ensemble des projections complétées à partir de π. Le contenu de cet ensemble est ajouté au résultat retourné par l'opération `OpeTargetedProjection` lorsque toutes les projections partielles de $\prod_{partiel}$ ont été parcourues.

6.4.2 Fermeture incrémentale d'un graphe par un ensemble de règles

La classe `OpeTargetedRulesClosure` permet de réaliser la fermeture incrémentale d'un graphe par un ensemble de règles $\mathcal{R}$. L'opération consiste à enchaîner successivement autant d'étapes d'applications possibles à partir des sommets produits par l'étape précédente. Une étape d'application consiste à exécuter `OpeApplyRule` sur chaque règle R de $\mathcal{R}$ selon chaque projection π_H de son hypothèse H dans la base. Une application de R selon π_H n'a lieu que s'il n'existe aucune projection π_C de la conclusion de R dans la base qui n'envoie chaque sommet frontière sur son image par π_H. La procédure de fermeture se termine lorsque l'étape courante n'a produit aucun nouveau sommet.

L'opération `OpeRuleApplications` est configurée pour utiliser l'opération `OpeTargeted-Projection` à la place de l'opération standard `OpeProjection`. Ce paramétrage permet de calculer la fermeture incrémentale à partir d'un sous-ensemble donné des sommets de la base d'assertions G ; comme spécifié pour la procédure d'enrichissement au chapitre précédent (section `Le service de raisonnement` $\mathcal{SG}$ sous-section 5.1.3).

[8] y est plus spécialisé que x ssi $y \in C_H^T$, $\tau_y \leq \tau_x$ et $m_y \leq m_x$ avec $l_G(x) = \tau_x : m_x$ et $l_H(y) = \tau_y : m_y$

6.4.3 Vérification incrémentale des contraintes

La vérification incrémentale de G par rapport à une contrainte C et un sous-ensemble E des sommets de G s'effectue à partir des opérations `OpeTargetedPositiveConstraintValidation` et `OpeTargetedNegativeConstraintVerification` selon la nature de la contrainte. `OpeTargetedProjection` est utilisée par ces deux opérations pour rechercher les projections qui ciblent les sommets spécifiés dans l'ensemble E.

Lorsque la contrainte est positive, la première opération retourne dans un objet `OpeResultProjection`, les projections de la partie condition de C sur au moins un sommet de l'ensemble E qui ne peuvent être étendues à la contrainte entière.

Lorsque la contrainte est négative, la seconde opération retourne dans le résultat `OpeResultProjection` les projections dans le graphe G de la contrainte négative C sur au moins un sommet de E.

6.5 Le système de validation et d'enrichissement

Le système de validation et d'enrichissement est composé de trois paquetages Java, le tout regroupant 256 classes Java[9].

Le paquetage de communication $\mathcal{SG}$ (nommé *std*) est dédié à la représentation Java de la famille $\mathcal{SG}$ et à la communication avec le serveur de raisonnement $\mathcal{SG}$. Ce paquetage envoie des directives au serveur, récupère les informations retournées (les échanges étant réalisés sous la forme de CoGXML encapsulé dans du SVECogitantCS) et les représente en Java sous la forme de graphes, règles ou contraintes $\mathcal{SG}$.

Les classes du service de validation et d'enrichissement d'ITM sont rangées dans le paquetage *itm*. Ce paquetage interprète les messages provenant d'ITM en instructions à fournir au paquetage de communication $\mathcal{SG}$, puis intègre les informations retournées par ce dernier pour qu'elles soient exploitables par ITM. L'initialisation du *SVE* est réalisée en chargeant l'ontologie, les règles et les contraintes associées à l'espace des annotations. Si l'espace des annotations n'est pas vide, la TM associée à cet espace est chargée après l'ontologie, les règles et les contraintes. Une fois le service initialisé, son exécution se déroule parallèlement à celle d'ITM. Chaque message d'alerte envoyé par ITM est analysé et conduit parfois à la recomposition d'une TM ajoutée. Le *SVE* déclenche alors la validation et l'enrichissement de cette TM en faisant appel au paquetage de communication $\mathcal{SG}$, puis intègre le résultat (connaissance inférée et violations).

Le paquetage de traduction (nommé *translators*) rassemble les classes qui permettent de traduire une structure de données du paquetage de validation et d'enrichissement (paquetage *itm*) en une structure de données du paquetage de communication $\mathcal{SG}$ (paquetage *std*) ; et inversement.

[9] 22600 lignes de code.

6.5.1 Paquetage de validation et d'enrichissement d'ITM

Le paquetage associé au *SVE* est composé de quatre sous-paquetages : les sous-paquetages de représentation d'une ontologie (*itm.ontology*) et du résultat d'une validation (*itm.controls*) et le sous-paquetage de validation d'une annotation.

Le sous-paquetage de représentation d'une ontologie est utilisé pour charger le niveau modèle en mémoire et le représenter sous la forme d'un objet Java ; cet objet étant voué, dans ce projet, à être traduit en une structure Java du paquetage de communication $\mathcal{SG}$ (représentant un support, des règles et des contraintes).

Le sous-paquetage de représentation d'un résultat de validation est utilisé pour représenter les violations de contraintes et la connaissance inférée sous la forme de TM dans la classe `ValidationResult`. La méthode `save()` de cette classe permet de sauvegarder les violations dans un fichier sur le disque et d'intégrer la TM inférée dans l'espace des annotations. Lorsqu'un objet `ValidationResult` est instancié, les violations sauvegardées sont chargées en mémoire. Les méthodes de la classe permettent de retrouver les annotations violant une contrainte donnée, les contraintes violées par une annotation donnée ou de parcourir l'ensemble de toutes les violations et de toutes les annotations.

Le sous-paquetage de validation d'une annotation regroupe la procédure d'initialisation (sous-paquetage *itm.validation.init*) et la procédure qui supervise l'analyse des messages d'alerte provenant d'ITM (dans *itm.validation.scheduler*). Cette analyse dépend de la nature de l'action à l'origine du message d'alerte reçu. Lorsque cette action est un ajout (resp. une suppression) la procédure d'analyse sollicitée est celle définie dans *itm.validation.scheduler.creation* (resp. *itm.validation.scheduler.deletion*). Dans les deux cas, l'analyse peut conduire au déclenchement d'une validation en utilisant la méthode `add` de la classe `ITMCogitantClient` qui retourne un objet `ValidationResult` contenant les violations et la connaissance inférée dues à l'ajout contrôlé de l'annotation TM.

6.5.2 Paquetage de communication $\mathcal{SG}$

Le paquetage de communication $\mathcal{SG}$ est divisé en trois sous-paquetages : Le sous-paquetage consacré à la représentation des composants de la famille $\mathcal{SG}$ (dans *std.cg*), le client en communication avec le serveur de raisonnement $\mathcal{SG}$ (*std.client*) et le sous-paquetage dédié à la traduction des documents XML reçus par le client en objets Java (*std.xml*).

Représentation de la famille $\mathcal{SG}$

La famille $\mathcal{SG}$ définit essentiellement quatre composants - les SG, les règles, les contraintes négatives et les contraintes positives - qui sont respectivement représentés en Java par les classes `ConceptualGraph`, `Rule`, `NegativeConstraint` et `Positive-`

`Constraint` (ces deux dernières étant sous-classes de `Constraint`). La figure 6.4 présente ces classes organisées selon la relation classe/sous-classe.

Les règles, les contraintes négatives et les contraintes positives sont considérées dans le formalisme $\mathcal{SG}$ comme des paires de graphes dont l'un est la partie hypothèse (ou condition) et l'autre la partie conclusion (ou obligation) ; la jonction entre les deux graphes étant représentée par des liens de connexion. Chaque paire de graphes et les liens de connexion sont représentés par un objet `ConceptualImplication`. Ce dernier est *composé* de deux objets `ConceptualGraph`, l'un représentant la partie hypothèse (ou condition) et l'autre la conclusion (ou obligation) ; ainsi que d'un objet `Connection-PointsList` qui rassemble les liens de connexion entre les deux graphes. Une contrainte ou une règle étant représentée par une paire de graphes et des liens de connexions, les classes associées `Rule` et `Constraint` étendent la classe `ConceptualImplication`.

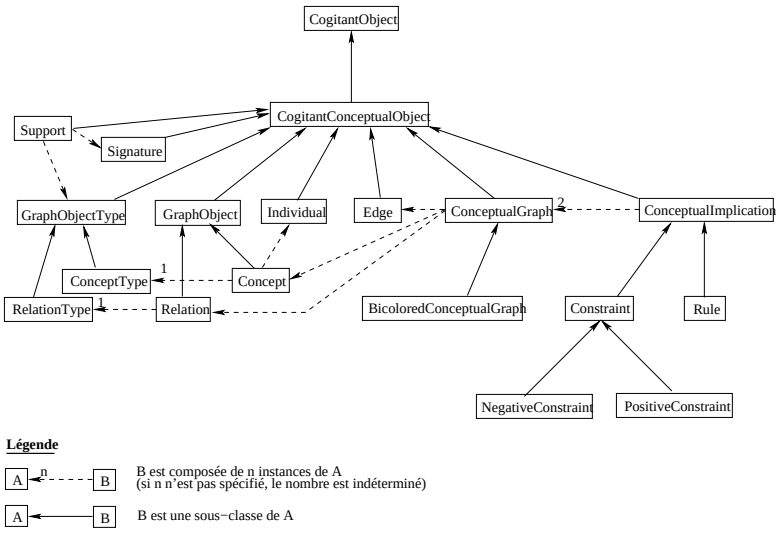


FIG. 6.4 – Classes modélisant le formalisme Topic Map.

Les règles et les contraintes peuvent aussi prendre la forme d'un SG bicoloré représenté en Java par la classe `BicoloredConceptualGraph`. Un SG bicoloré étant un SG muni d'une fonction de coloration, cette classe étend la classe `ConceptualGraph`. Les classes `BicoloredConceptualGraph` et `ConceptualImplication` disposent chacune d'une méthode réalisant la traduction de l'une en l'autre.

Un objet `ConceptualGraph` est *composé* d'objets `Concept`, `Relation` et `Edge` ; les deux premiers étant sous-classes de `GraphObject`. Chaque objet `Concept` ou `Relation` fait référence à un objet `GraphObjectType` respectivement `ConceptType` ou `Relation-Type` (représentant les types de ces sommets) ; les premiers pouvant en plus posséder

un objet `Individual` (représentant le marqueur individuel du sommet concept). L'objet `Concept` détenteur d'un objet `Individual` de valeur *null* représente un sommet concept générique.

La classe `GraphObjectType` dispose de méthodes pour savoir si un objet `GraphObjectType` passé en paramètre représente un supertype ou un sous-type du type représenté par l'objet détenteur de la méthode (`isSubTypeOf`, `isSuperTypeOf`), ou pour connaître l'ensemble des supertypes ou sous-types du type représenté par l'objet considéré (`isSubTypeOf`, `getSubTypes`, `setSubTypes`, `isSuperTypeOf`, `getSuperType`, `setSuperType`).

L'objet `Support` rassemble des objets `ConceptType`, `RelationType` ou `Individual`.

Toutes ces classes étendent la classe `CogitantConceptualObject` ; elle même sous-classe de `CogitantObject`.

Le client

Le client se structure en deux niveaux : le client $\mathcal{SG}$ et le client réseau. Les méthodes du client $\mathcal{SG}$ prennent en argument ou retournent comme résultat des objets Java correspondant aux composants de la famille $\mathcal{SG}$. Celles du client réseau prennent en argument et retournent des documents XML (aux formats CoGXML et SVECogitantCS) sous la forme de chaînes de caractères. Le client $\mathcal{SG}$ fait appel aux méthodes du client réseau ; ce dernier étant en communication TCP/IP avec le serveur de raisonnement $\mathcal{SG}$. Les deux clients sont rangés dans le paquetage *std.client*.

Le client $\mathcal{SG}$

Le client $\mathcal{SG}$ (classe `StandardCogitantClient`) dispose essentiellement des trois méthodes `initialisation()`, `add()` et `query()`.

`initialisation()` prend en paramètre le nom d'un fichier contenant un support, des règles et des contraintes ; traduit le contenu de ce fichier en un objet `CogitantCSDocument` qui est ensuite transformé en chaîne de caractères et passé en argument de la méthode `send()` du client réseau.

`add()` et `query()` prennent en paramètre un objet `ConceptualGraph` et le traduisent en une chaîne de caractères (CoGXML encapsulé dans des directives SVECogitantCS) ; cette dernière étant passée en argument de la méthode `send()` du client réseau. Puis, elles traduisent la chaîne de caractères retournée par `send()` en un objet `CogitantCSDocument`.

Le client réseau

Le client réseau (classe `BasicCogitantClient`) fournit la méthode `send()` utilisée par le client $\mathcal{SG}$. Cette méthode envoie la chaîne de caractères passée en paramètre à

travers le réseau vers le serveur de raisonnement $\mathcal{SG}$ auquel il est connecté en liaison TCP/IP. Elle retourne ensuite sous forme de chaîne de caractères les informations reçues à partir du serveur (document SVECogitantCS).

Traitement des formats XML

Les formats CoGXML et SVECogitantCS permettent respectivement de représenter en XML (1) les graphes, les règles, les contraintes et le support de la famille $\mathcal{SG}$ et (2) les opérations que le serveur doit exécuter. En plus de l'être par le client $\mathcal{SG}$ pour communiquer avec le serveur de raisonnement, ces formats sont utilisés pour la sauvegarde sur disque de connaissances $\mathcal{SG}$. Les objets Java `CoGXMLDocument` et `CogitantCSDocument` permettent de représenter en Java les documents CoGXML et SVECogitantCS. Une transformation (un *parser* XML) est définie pour traduire une chaîne de caractères respectant le format CoGXML en un objet `CoGXMLDocument` ou `CogitantCSDocument` et inversement. Cette transformation a été conçue à l'aide de l'outil *SAX* (voir [Meg98]) qui permet de traiter les documents XML en flux continu (et évite de les charger tout entiers en mémoire). Ces transformations sont définies dans le paquetage *std.xml*.

Pour traduire un document au format CoGXML (resp. SVECogitantCS) en l'objet Java correspondant, il faut faire appel aux méthodes `parse` de la classe `CoGXMLDocument` (resp. `CogitantCSDocument`). Ces méthodes retournent un objet `CoGXMLDocument` (resp. `CogitantCSDocument`) et prennent en argument la chaîne ou le flux de caractères qui contient le document.

Pour réaliser la traduction inverse, il faut appeler la méthode `cogxml` de la classe `CoGXMLDocument` qui retourne le document XML dans une chaîne de caractères au format CoGXML.

6.5.3 Paquetage de traduction

Le paquetage de traduction dans *translators* implémente les transformations présentées au chapitre 4. Il prend en charge la traduction f_{TM2S} d'une ontologie TM en un support ; les traductions f_{MIN2C^+} et f_{MAX2C^-} des cardinalités maximales et minimales en contraintes négatives et positives ; la traduction f_{H2RC^-} des propriétés hiérarchiques en règles et contraintes ; et enfin la traduction f_{TM2SG} d'une annotation TM en un SG et d'un SG en TM (par f_{SG2TM}). D'un point de vue général, ces transformations permettent de traduire une structure de données du paquetage de validation et d'enrichissement d'ITM (*itm*) en une structure de données du paquetage de communication $\mathcal{SG}$ (*std*) (et inversement).

Export d'une ontologie TM en ontologie SG

La transformation d'une ontologie TM en une ontologie SG consiste à charger la TM du niveau modèle dans un objet `TopicMapOntology` et à traduire ce dernier en un objet `Support` et un ensemble d'objets `PositiveConstraint`, `NegativeConstraint` et `Rule`. Cette transformation est définie dans le paquetage *translators.tmo2s*. Elle est réalisée en quatre étapes par la méthode `translate()` de la classe `TopicMapOntology2CoGXMLDocument`. Ces quatre étapes correspondent aux applications successives des fonctions f_{TM2S}, f_{MIN2C^+}, f_{MAX2C^-} et f_{H2RC^-} sur la TM de l'espace ontologique.

La transformation f_{TM2S} est réalisée par la classe `TopicMapOntology2Support`. Cette dernière fait appel à la classe `TopicMapTypes2SupportTypes` pour traduire les types et classes de l'ontologie TM en les hierarchies de types de concepts et types de relations du support. Les transformations f_{MIN2C^+} et f_{MAX2C^-} sont respectivement implémentées par les classes `MinimumCardinalities2PositiveConstraints` et `MaximumCardinalities2NegativeConstraints`. Enfin, f_{H2RC^-} est réalisée par la classe `TopicMapHierarchicalConstraints2RulesAndConstraints` qui fait appel à `Hierarchical2NegativeConstraint` et `Hierarchical2Rule` ; générant respectivement une contrainte négative et une règle pour chaque association hiérarchique de l'ontologie.

Traduction d'une annotation TM en SG (fonction f_{TM2SG})

La transformation d'une TM d'un espace de travail en SG est définie dans le sous-paquetage *tm2cg*. La classe `TopicMap2ConceptualGraph` implémente la fonction f_{TM2SG} et s'exécute en deux temps.

La première étape consiste à traduire chaque topic et ses occurrences en le graphe conceptuel "étoile" composé, au centre, du sommet concept correspondant au topic ; et en périphérie, des sommets concepts correspondant aux valeurs des occurrences. Les sommets relations correspondent aux types logiques des occurrences et relient le sommet concept central aux sommets concepts périphériques. La traduction d'un topic est réalisée par la classe `Topic2ConceptualGraph`.

La seconde étape consiste à traduire chaque association et ses occurrences en un graphe conceptuel étoile dont le sommet concept au centre est celui correspondant à l'association et ceux en périphérie correspondent aux valeurs d'occurrences et aux topics ; quant aux relations elles correspondent soit aux rôles, soit aux types logiques des occurrences. La traduction d'une association est implémentée par la classe `Association2ConceptualGraph`.

Les graphes issus de la traduction des topics et des associations sont ensuite fusionnés en un seul SG qui est le résultat de la traduction de la TM.

Export de l'espace des annotations sous la forme d'un SG (fonction $f_{TMKB2SG}$)

Lorsque l'espace des annotations n'est pas vide et qu'il est nécessaire d'initialiser le *SVE*, le contenu de cet espace est traduit en un SG qui est ajouté par ajout contrôlé dans $B_{\mathcal{SG}}$. La traduction de l'espace ne peut être réalisée par la transformation précédente car le résultat de cette transformation étant toujours chargé en mémoire, son application sur $tm_{\mathcal{A}}$ (généralement très volumineuse) peut conduire à une saturation de la mémoire et une interruption du programme. Dans ce cas, il faut recourir à l'export SG de l'espace des annotations qui consiste à traduire par petites portions la TM de l'espace et à les sauvegarder progressivement en CoGXML dans un fichier sur le disque.

Traduction d'une annotation SG en TM (fonction f_{SG2TM})

La transformation f_{SG2TM} inverse de f_{TM2SG} est définie dans le sous-paquetage *cg2tm* et réalisée par la classe `ConceptualGraph2TopicMap` en deux étapes.

La première consiste à repérer dans l'objet `ConceptualGraph` les motifs identifiant les topics et leurs occurrences (tâche réalisée par `ConceptualGraph2TopicPatterns`) ; puis traduire chacun de ces motifs en topics (en faisant appel à `TopicPatterns2Topics`).

La seconde applique le même principe pour obtenir les associations : chaque motif identifiant une association (trouvé à l'aide de `ConceptualGraph2AssociationPatterns`) est traduit en une association (avec la classe `AssociationPatterns2Associations`).

Les topics et associations obtenus sont rassemblés dans un objet `TopicMap` qui correspond au résultat de la transformation.

6.5.4 Expérimentations et mesures

Un paquetage d'expérimentation a été développé pour évaluer les performances du service de validation et d'enrichissement. Ce paquetage fonctionne en deux étapes.

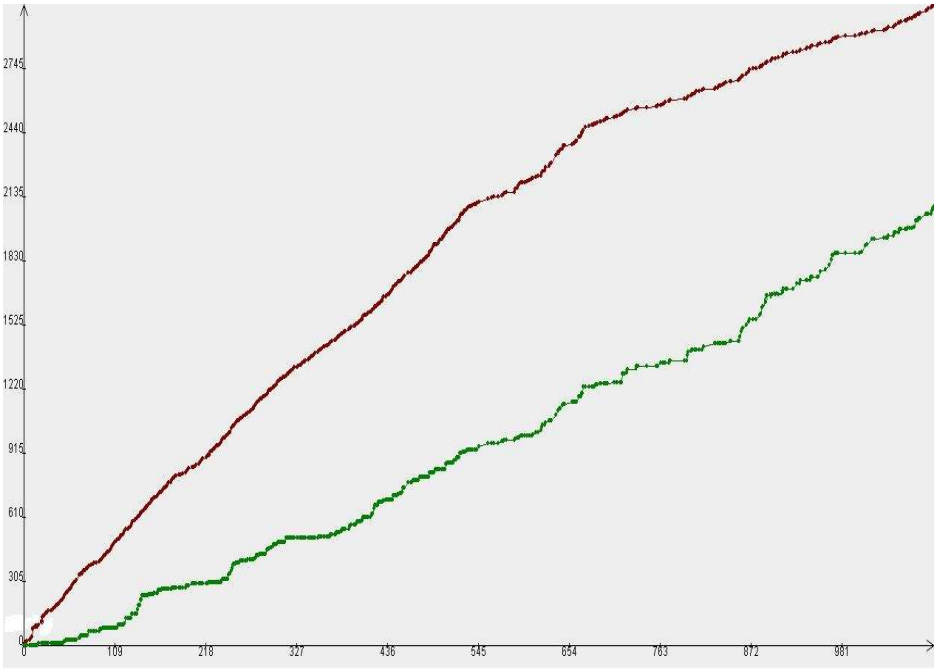


FIG. 6.5 – Le nombre de relations assertées (courbe du haut) et inférées (courbe du bas) dans la base du serveur de raisonnement $\mathcal{SG}$ après l'ajout contrôlé de la x^{ieme} annotation.

La première réalisée par le sous-paquetage *stats.model* consiste à archiver sur le disque les données échangées entre les différents processus du programme à évaluer. En plus du contenu des données, la procédure d'archivage sauvegarde le moment de capture de la donnée, le processus dont elle est le résultat et celui auquel elle est passée en argument.

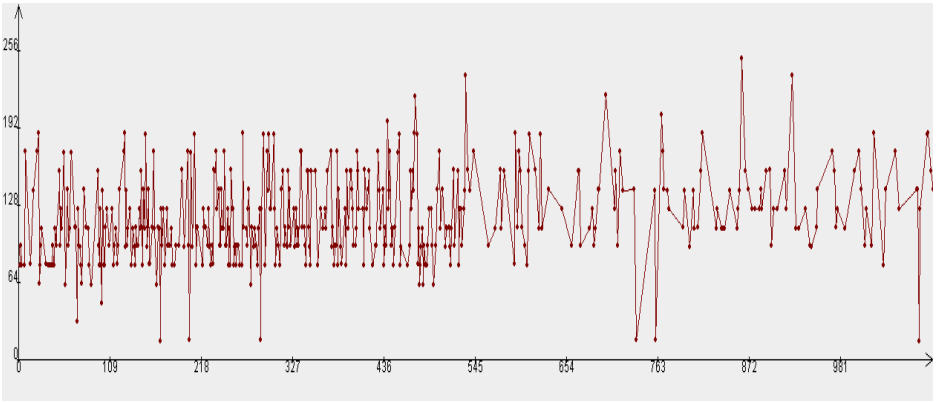


FIG. 6.6 – Le temps de traitement en ms de la x^{ieme} annotation reçue par le système d'inférence.

La seconde étape consiste à parcourir l'ensemble des archives pour réaliser des mesures. Le profil des informations recherchées dans les fichiers est défini à l'aide des outils fournis par le paquetage *stats.criterias* et à chaque profil est associée une opération définie à partir des classes de *stats.formulas*. Un paquetage de calcul se charge de réaliser l'opération associée au profil de chaque information trouvée et le paquetage de visualisation graphique défini dans *stats.graphics* présente les résultats sous la forme de courbes.

Ce paquetage a été utilisé dans le cadre du projet PressIndex pour évaluer la quantité de nouvelles connaissances inférées au cours de la phase d'enrichissement et les performances du serveur de raisonnement $\mathcal{SG}$ en montée de charge sur des données métiers. Les expérimentations réalisées ont porté sur douze mille documents. L'extraction linguistique a duré près de 6 heures et a conduit à l'extraction de plus de mille annotations et à la création de trois mille topics reliés entre eux par mille associations.

La base de connaissances du serveur de raisonnement $\mathcal{SG}$ contenait 50 règles et 76 contraintes à l'initialisation. A la fin du processus, elle comptait approximativement cinq mille sommets concepts et un nombre de sommets relations légèrement inférieur. Les courbes des figures 6.6 et 6.5 présentent respectivement le temps de traitement du service de validation et d'enrichissement et le nombre de relations assertées (courbe du dessus, fig 6.5) ou inférées (courbe du bas, même figure) au cours de l'ajout contrôlé de la x-ème annotation. Pour une même annotation, le temps de traitement reste relativement constant (entre 10 et 200 ms à la figure 6.6) même quand la charge augmente (jusqu'à 5000 sommets concepts et 4000 sommets relations comme l'illustre la figure 6.5). De plus, on remarque que le nombre de relations inférées augmente en fonction du nombre de relations assertées ; ce qui confirme l'utilité de la phase d'enrichissement.

Le paquetage d'expérimentations met aussi à disposition un générateur de graphes

conceptuels (dans *stats.generator*). Le programme retourne le nombre spécifié de graphes conceptuels ; les quantités de sommets, d'individuels et d'arêtes de ces derniers étant choisis aléatoirement entre des valeurs fixées. Ce générateur a été utilisé pour tester la résistance à la charge du serveur de raisonnement $\mathcal{SG}$. Le test consistait à générer un SG de 1.200.000 sommets concepts (700.000 individuels et 500.000 génériques) et 1.700.000 sommets relations puis à le charger dans le serveur. Pour un tel graphe, le chargement en mémoire a duré approximativement 5 minutes et l'espace mémoire alloué était de 850 Mo (la machine utilisée était un portable à prix raisonnable de l'année 2005).

Bien que d'autres expérimentations soient à prévoir pour précisément évaluer la robustesse et la qualité des traitements réalisés par le service de raisonnement $\mathcal{SG}$, les résultats actuels sont suffisamment positifs pour envisager de continuer et terminer le développement du service de validation et d'enrichissement d'ITM.

6.5.5 Conclusion

Ce chapitre présentait la manière dont le service de validation et d'enrichissement d'annotations (*SVE*) a été implémentée dans ITM. En résumé, le *SVE* est divisé en trois grandes parties : la *partie TM* qui interagit avec ITM, la *partie SG* qui réalise les raisonnement et la *partie intermédiaire* qui effectue les traductions entre les objets correspondant au langage TM et ceux correspondant à la famille $\mathcal{SG}$. Bien que le *SVE* ait été conçu à l'origine pour ITM, on peut remarquer que seule la partie TM propose des services exclusivement réservés à ITM. En effet, le *SVE* s'appuie sur des modules relativement génériques qui peuvent être réutilisés pour concevoir d'autres applications :

- On considère l'application SVE_{TM} limitée aux deux seules parties *intermédiaires* et $\mathcal{SG}$ du *SVE*. Un outil de gestion des connaissances qui emploie le langage TM utilisé dans ITM (i.e. respectant les conditions énoncées dans le chapitre `Les Topic Maps`) peut exploiter les services de *validation* et d'*enrichissement* fournis par la partie *intermédiaire* de SVE_{TM}.
- Une application $SVE_{\mathcal{SG}}$ créée à partir de la *partie* $\mathcal{SG}$ du *SVE* permet de valider et d'enrichir des annotations définies à partir de la famille $\mathcal{SG}$. $SVE_{\mathcal{SG}}$ conserve l'avantage d'être une application client-serveur, les opérations coûteuses étant réalisées par le serveur (le service de raisonnement $\mathcal{SG}$ développé en C++) sans pénaliser le client (développé en Java).

Chapitre 7

Conclusion

Les travaux présentés dans cette thèse ont permis de doter un outil de gestion des connaissances basé sur le langage des TM de capacités de raisonnement. Pour ce faire, il a fallu définir une série de transformations du langage des TM vers un formalisme de représentation des connaissances muni d'une sémantique formelle : la famille $\mathcal{SG}$ des graphes conceptuels.

Pour résumer, le cœur de cet outil - baptisé ITM et développé par la société Mondeca - est une base de connaissances structurée en 3 niveaux :

1. Le niveau méta contient l'ontologie de représentation.
2. Au niveau modèle, des ontologies sont créées pour chaque client en utilisant les primitives de l'ontologie de représentation.
3. Le niveau des instances contient des annotations sémantiques, thésaurus, et des descriptions organisationnelles construites en utilisant le vocabulaire conceptuel des ontologies du niveau modèle.

Le langage de représentation utilisé dans ITM est basé sur les "Topic Maps" (TM) ; un formalisme pouvant être modélisé par des graphes dont les sommets sont les topics et associations, et les arêtes indiquent la participation d'un topic à une association. Le langage des TM dispose d'une syntaxe XML qui en fait un des langages candidats à une utilisation dans le cadre du web sémantique. Contrairement à ses concurrents directs, RDF/S et OWL, les TM ne sont pas munies d'une sémantique formelle. Aucun outil complet de raisonnement ne leur est dédié.

La famille $\mathcal{SG}$ est un formalisme de graphes conceptuels permettant la représentation de différents types de connaissances sous la forme de supports (sorte d'ontologie rudimentaire), de SG, de règles ou de contraintes ; ces représentations possédant toutes une sémantique formelle en logique des prédicats du premier ordre. La famille $\mathcal{SG}$ offre différents mécanismes de raisonnement variant de la saturation d'un graphe par rapport à un ensemble de règles, à la satisfaction d'un ensemble de contraintes par ce dernier.

Ces mécanismes sont tous fondés sur un homomorphisme de graphes appelé "projection" (issu de la théorie des graphes) et sont adéquats et complets vis à vis de la déduction en logique des prédicats.

Interopérabilité des langages

Les proximités qu'entretient le formalisme des TM avec les formalismes de graphes conceptuels et en particulier la famille $\mathcal{SG}$, ont conduit à plonger les représentations d'ITM dans cette famille de langages. Une série de transformations permettent de traduire la portion adéquate du niveau modèle en un support, un jeu de règles et un jeu de contraintes. Deux transformations ont été définies, l'une traduisant une TM du niveau instance en un SG et l'autre réalisant la traduction inverse. Ces transformations étant injectives, elles attribuent la sémantique formelle de la famille $\mathcal{SG}$ aux représentations d'ITM et ont conduit à définir un service de validation et d'enrichissement d'annotations pour munir l'outil ITM de mécanismes de contrôle et de raisonnement.

Raisonnement et qualification des connaissances

Le service de validation et d'enrichissement (SVE) fonctionne en quatre étapes et consiste à *contrôler*, *intégrer*, *enrichir* et *valider* l'annotation reçue. Il maintient une copie, enrichie par application de règles d'inférence, d'une portion "cohérente" de la base d'annotations ITM. Le choix de la portion "cohérente" est déterminé par l'ordre d'ajout des annotations à la base ; toute nouvelle annotation rendant la base incohérente étant rejetée. Ceci permet d'assurer la monotonie des raisonnements effectués sur cette base, sans pour autant perdre les connaissances rejetées puisqu'elles sont conservées dans ITM. Le résultat retourné par le service de validation et d'enrichissement à ITM contient éventuellement les connaissances inférées à partir de l'annotation et les violations auxquelles elle a pu conduire. L'utilisateur peut ensuite améliorer l'annotation acceptée qui présente certains manques, ou la réparer si elle a été rejetée.

La solution est construite selon une approche *incrémentale* : d'une part, chaque résultat retourné par le SVE ne concerne que l'ajout courant de connaissances et d'autre part, les étapes de *contrôle*, d'*intégration*, d'*enrichissement* et de *validation* ont été conçues pour cibler la portion des connaissances de la base provenant de l'ajout incrémental. Le caractère incrémental de l'approche garantit un temps de réponse raisonnable pour une quantité importante d'informations dans la base.

Le SVE a été implémenté dans une application élaborée à partir de la plate-forme CoGITaNT permettant le développement d'applications basées sur le modèle des graphes conceptuels. L'application se présente sous la forme d'un client-serveur dont le serveur héberge le service de raisonnement $\mathcal{SG}$ et le client - à la fois en communication avec ITM et le serveur - orchestre la validation et l'enrichissement des annotations ajoutées

à ITM. Cette architecture client-serveur permet d'exécuter le service de raisonnement $\mathcal{SG}$ sur une machine distante de celle qui héberge ITM ce qui offre l'avantage de réaliser des traitements éventuellement coûteux, sans pour autant faire chuter les performances d'ITM.

Portée de la solution

Le SVE se présente sous la forme d'une API externe à ITM. Il a la particularité d'être modulable ; ce qui donne la possibilité de le décomposer en deux modules SVE_{TM} et $SVE_{\mathcal{SG}}$ qui peuvent être utilisés pour munir d'autres outils de capacités de raisonnement. SVE_{TM} convient à un outil de gestion des connaissances autre qu'ITM et toujours basé sur la portion traduite des TM ; à condition que ce dernier respecte la structuration du langage des TM en trois niveaux méta/modèle/instances et que son interprétation d'une TM soit identique à celle d'ITM. $SVE_{\mathcal{SG}}$ peut être "branché" sur un outil utilisant un autre langage à condition de redéfinir une transformation de ce langage vers la famille $\mathcal{SG}$.

Le SVE a été exploité dans deux projets : le projet PressIndex (application dédiée à la veille médiatique) et le projet Eiffel (application dédiée au domaine du tourisme). Ces projets ont permis de tester la solution, d'en apprécier les limites et, en vue de l'améliorer, de dresser quelques perspectives de recherche présentées dans la section suivante.

Perspectives

Le travail réalisé ici a permis de répondre à certains besoins d'ITM en matière de *raisonnement* et *qualification* de ses connaissances. Cette solution peut encore être améliorée. Les réflexions menées au cours de ce travail se prolongent principalement en quatre perspectives de recherche. La première concerne l'amélioration du mécanisme de maintenance de la base, la seconde la prise en compte des suppressions dans la portion des connaissances acceptées de l'espace des annotations, la troisième la prise en charge d'opérateurs de comparaison réservés aux valeurs d'occurrences et la quatrième la définition d'une architecture logicielle générique permettant de raisonner avec la famille $\mathcal{SG}$ sur le langage des TM.

Maintenance de la base.

En plus des annotations, le service de validation et d'enrichissement permet l'ajout contrôlé de règles ou de contraintes. Cependant lorsque cette règle ou cette contrainte est rejetée, les raisons de son rejet ne sont pas décrites dans l'espace des violations. La description de ces rejets permettrait de fournir à l'utilisateur davantage d'informations pour modifier la base de telle sorte que la règle ou la contrainte rejetée soit acceptée. D'une manière générale, la description des violations pourrait être améliorée en utilisant

les projections non plus seulement pour identifier les violations mais aussi pour repérer les sommets topics ou association qui sont à l'origine d'un excès ou d'un manque de connaissance. En effet, si chaque contrainte est exprimée sous la forme d'un graphe bicoloré (même les contraintes négatives), elle dispose de sommets frontières dont l'image par projection constitue une information pouvant faciliter la compréhension de la violation et, par là même, l'amélioration ou la réparation de l'annotation concernée.

Suppression de connaissances.

Lorsqu'une suppression a lieu parmi la connaissance acceptée de l'espace des annotations, elle entraîne une désynchronisation entre ITM et le service de raisonnement $\mathcal{SG}$; ce dernier n'ayant pas la faculté de gérer la suppression à la volée d'un sous-graphe dans sa base tout en la conservant saturée par rapport aux règles. Si la suppression est absolument nécessaire, la solution actuelle consiste à redémarrer le service de raisonnement, à recharger la base après suppression et à relancer la saturation sur son intégralité ; ce qui peut être coûteux lorsque la base est volumineuse. Un algorithme dit *incrémental* de suppression a donc été conçu pour réaliser efficacement de telles suppressions.

L'algorithme permet de supprimer un sous-graphe H du graphe saturé en entrée tout en conservant le graphe résultant saturé par rapport aux règles. L'algorithme fonctionne en deux étapes.

La première étape supprime l'ensemble des sommets produits par une saturation incrémentale démarrant à partir de H dans le graphe donné. Seulement, cette étape a pu supprimer des sommets qu'il faut conserver : les sommets provenant de la saturation réalisée à partir de H qui sont inférés à partir d'un sous-graphe disjoint de H.

La seconde étape prend ce cas en considération et relance une saturation *incrémentale* à partir des sommets adjacents au sous-graphe supprimé (c'est à dire les sommets voisins des sommets supprimés) ce qui permet de rajouter les éventuels sommets qui manquent au graphe résultant de la suppression pour être saturé par rapport aux règles.

Une fois ces étapes achevées, l'algorithme détecte et retourne les contraintes positives violées ou réparées par la suppression. Comme le graphe de la base avant suppression respecte les contraintes négatives, il en est de même pour celui résultant de la suppression, puisque ce dernier est toujours plus général que le premier (l'opération de suppression est généralisante). Il reste à prouver l'algorithme et à justifier l'intérêt de la méthode en démontrant que le nombre de sommets supprimés qu'il fallait en fait conserver est le plus souvent "petit".

Valeurs d'occurrences.

Les occurrences de ITM sont représentées par des attributs numériques ou textuels dans un SGBD qui fournit des opérateurs de comparaison adaptés pour les interroger : requêtes du type "retourne tous les topics dont la valeur d'une occurrence de type en-

tier est comprise entre deux nombres" ou "retourne tous les topics dont le nom contient telle sous-chaîne". Comme énoncé dans le chapitre 3 sur les transformations, la représentation de ces valeurs dans un SG est possible bien qu'elle ne soit pas d'une grande utilité. En effet, les valeurs d'occurrences n'ont pas la même sémantique que les marqueurs individuels des sommets concepts. Elles peuvent désigner des entiers, des dates ou des chaînes de caractères. Leur taille en mémoire est variable et elles utilisent très peu les avantages d'une représentation réseau puisqu'étant souvent différentes, elles sont représentées par un ajout de sommets pendants. Une représentation hybride SGBD/GC semble être un bon compromis pour obtenir un formalisme de représentation des connaissances qui conjugue les avantages d'une modélisation par attribut-valeur à ceux d'une représentation par graphes conceptuels tout en exploitant les capacités d'interrogation du SGBD sur les valeurs.

Transformations.

Les transformations établies du langage des TM vers celui de la famille $\mathcal{SG}$ ont permis d'*opérationaliser* la portion restreinte aux TM n'utilisant pas les domaines de validité et les variations de nom, de la doter d'une sémantique formelle (puisque les transformations sont réversibles) ainsi que de capacités d'inférences et de contrôle à partir de règles et de contraintes définies dans la famille $\mathcal{SG}$. L'amélioration de ces transformations afin de traduire les éléments manquants du langage TM conduirait à une solution pour raisonner sur la totalité du langage des TM.

Bibliographie

[13206] ISO 13250 :2006. Xml topic maps (xtm) 2.0 - final draft. http ://www.isotopicmaps.org/sam/sam-xtm/2006-06-19/, 2006.

[90706] ISO/IEC 9075 :2003. Information technology - database languages - sql. 2006.

[AdMRV02] P. Auillans, P. Ossona de Mendez, P. Rosenstiehl, and B. Vatant. A formal model for topic maps. In *Proc. of ISWC'02*, pages 69–83, 2002.

[ALM05] F. Amardeilh, P. Laublet, and J.L. Minel. Annotation documentaire et peuplement d'ontologie à partir d'extractions linguistiques. In *Actes IC*, 2005.

[Ama06] F. Amardeilh. Ontopop or how to annotate documents and populate ontologies from texts. In *Proc. of the ESWC 2006 Workshop on Mastering the Gap : From Information Extraction to Semantic Representation*. CEUR Workshop Proceedings, 2006. http ://ceur-ws.org/Vol-187/6.pdf.

[Baa96] F. Baader. Logik-basierte wissensrepresentation (extended and updated english version). *KI*, 10(3) :8–16, 1996.

[Bac04] B. Bachimont. Pourquoi n'y a-t-il pas d'expérience en ingénierie des connaissances ? *IC 2004 : 15es journées francophones d'ingénierie des connaissances, Lyon, France*, pages 53–64, 2004.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press, 2003.

[Bec04] D. Beckett. Rdf/xml syntax specification, recommendation w3c. http ://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, 2004.

[Ber58] C. Berge. *Théorie des graphes et ses applications*. Dunod, Paris, 1958.

[BFP98] V.R. Benjamins, D. Fensel, and A.G. Perez. Knowledge Management through Ontologies. *PAKM 98 Practical Aspects of Knowledge Management-Proceedings of the Second International Conference*, 1998.

[BG04] D. Brickley and R.V. Guha. Rdf vocabulary description language 1.0 : Rdf schema. http ://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 2004.

[BL88] T. Berners-Lee. Cern experience. In *ACM SIGOPS European Workshop*, 1988.

[BLCG92] T. Berners-Lee, R. Cailliau, and J.-F. Groff. The world-wide web. *Computer Networks and ISDN Systems*, 25(4-5) :454–459, 1992.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. 2001.

[BM02] J.-F. Baget and M.-L. Mugnier. Extensions of Simple Conceptual Graphs : The Complexity of Rules and Constraints. *JAIR*, 16 :425–465, 2002.

[BN01] M. Biezunski and S. R. Newcomb. A processing model for xml topic maps. http ://www.topicmaps.net/pmtm4.htm, 2001.

[Boi01a] B. Boiko. *Content Management Bible*. John Wiley & Sons, Inc. New York, NY, USA, 2001.

[Boi01b] B. Boiko. Understanding Content Management. *Bulletin of the American Society for Information Science and Technology*, 28(1) :8–13, 2001.

[BS05] R. A. Barta and G. Salzer. The tau model, formalizing topic maps. In *Proc. of APCCM'05*, pages 37–42, 2005.

[BS06] J.-F. Baget and E. Salvat. Rules dependencies in backward chaining of conceptual graphs rules. In *ICCS*, pages 102–116, 2006.

[CBT05] J. Charlet, B. Bachimont, and R. Troncy. Ontologies pour le web sémantique. *Le Web Sémantique (J. Charlet, P. Laublet, C. Reynaud)*, pages 43–63, December 2005.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1, note w3c. http ://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[Cha02] J. Charlet. *L'ingénierie des connaissances : développements, résultats et perspectives pour la gestion des connaissances médicales*. PhD thesis, Université Paris 6, 2002.

[Cha05] J. Charlet. L'ingénierie des connaissances, une science de gestion ? *Entre la connaissance et l'organisation, l'activité collective, chapitre 11. La découverte, 2005. Actes du colloque de Cerisy « Activité, connaissance, organisation »*., 2005.

[CM92] M. Chein and M.-L. Mugnier. Conceptual Graphs : Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4) :365–406, 1992.

[DNB06] P. Durusau, S. Newcomb, and R. Barta. Topic maps - reference model : Iso 13250-5 draft. http ://www.isotopicmaps.org/TMRM/TMRM-6.0/TMRM-6.0.pdf, 2006.

[eA] Open RDF et Aduna. Sesame. http ://www.openrdf.org/.

[Fur04] F. Furst. Toocom : Tool to operationalize an ontology with the conceptual graph model. http ://sourceforge.net/projects/toocom, 2004.

[Gar03] L. Marius Garshol. Living with topic maps and rdf. http ://www.ontopia.net/topicmaps/materials/tmrdf.html, 2003.

[Gar05a] L. M. Garshol. Q : A model for topic maps : Unifying rdf and topic maps. In *Proc. of the Extreme Markup Languages Conference*, 2005.

[Gar05b] L. M. Garshol. tolog - a topic maps query language. In *Proc. of TMRA'05*, pages 183–196, 2005.

[Gen97] D. Genest. Cogitant (conceptual graphs integrated tools allowing nested typed graphs). http ://cogitant.sourceforge.net, 1997.

[GM02] R. Gentilucci and P. Marco. Metainformazioni sul world wide web : Conversione di formato e navigazione (english version). Master Thesis, University of Bologna, 2002.

[GM06] L. Marius Garshol and G. Moore. Topic maps - data model : Iso 13250-2. http ://www.isotopicmaps.org/sam/, 2006.

[Gua98] N. Guarino. Formal ontology in information systems. pages 3–15. Amsterdam, IOS Press, 1998.

[Gut06] A. Gutierrez. Cogui (conceptual graphs graphical user interface) : A user friendly tool for building knowledge base. http ://www.lirmm.fr/cogui, 2006.

[Hay04] P. Hayes. Rdf semantics, w3c recommendation. http ://www.w3.org/TR/rdf-mt/, 2004.

[HM01] V. Haarslev and R. M"oller. Description of the racer system and its applications. In *Description Logics*, 2001.

[HP02] S. Hunting and J. Park. *XML Topic Maps : Creating and Using Topic Maps for the Web*. Addison-Wesley, 2002.

[HR05] M.-S. Hacid and C. Reynaud. L'intégration de sources de données. *Le Web Sémantique (J. Charlet, P. Laublet, C. Reynaud)*, pages 65–78, December 2005.

[ISO00] ISO/IEC :13250. Topic maps : Document description and markup languages. http ://www.y12.doe.gov/sgml/sc34/document/0129.pdf, 2000.

[KC04] G. Klyne and J. J. Carroll. Resource description framework (rdf) : Concepts and abstract syntax. http ://www.w3.org/TR/rdf-concepts/, 2004.

[KPS+06] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca Grau, and J. A. Hendler. Swoop : A web ontology editing browser. *J. Web Sem.*, 4(2) :144–153, 2006.

[KT05] P. Kellert and F. Toumani. Les web services sémantiques. *Le Web Sémantique (J. Charlet, P. Laublet, C. Reynaud)*, pages 93–110, December 2005.

[LD01] M. S. Lacher and S. Decker. On the integration of topic maps and rdf data. In *Proc. of the Extreme Markup Languages Conference*, 2001.

[LM06] M. Leclère and M.-L. Mugnier. Simple Conceptual Graphs with Atomic Negation and Difference. In *Proc. of ICCS'06*, pages 331–345, 2006.

[MB08] A. Miles and S. Bechhofer. Skos simple knowledge organization system reference, proposition w3c. http ://www.w3.org/TR/2008/WD-skos-reference-20080125/, 2008.

[Meg98] D. Megginson. Sax (simple api for xml). http ://www.saxproject.org, 1998.

[Moo01] G. Moore. Rdf and topic maps : An exercise in convergence. *Proc. of the XML Europe Conference*, 2001.

[MvH04] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview, recommendation w3c. http ://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.

[Ogi01] N. Ogievetsky. Xml topic maps through rdf glasses. *Markup Lang.*, 3(3) :333–364, 2001.

[Ora] Oracle. The oracle 11g rdf database. http ://www.oracle.com/technology/tech/semantic technologies/index.html.

[PG05] Y. Prié and S. Garlatti. Annotations et métadonnées dans le web sémantique. *Le Web Sémantique (J. Charlet, P. Laublet, C. Reynaud)*, pages 25–41, December 2005.

[PSHH04] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax, w3c recommendation. http ://www.w3.org/TR/owl-semantics/, 2004.

[PVG+06] S. Pepper, F. Vitali, L. Marius Garshol, N. Gessa, and V. Presutti. A survey of rdf/topic maps interoperability proposals. W3C Working Group Note - http ://www.w3.org/TR/2005/WD-rdftm-survey-20050329, 2006.

[SBA+02] A. Sheth, C. Bertram, D. Avant, B. Hammond, K. Kochut, Y. Warke, and I. Voquette. Semantic Content Management for Enterprises and the Web. *IEEE Internet Computing*, 6(4) :80–87, 2002.

[SM96] E. Salvat and M.-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *Proc. of ICCS'96*, volume 1115 of *LNAI*, pages 248–262. Springer, 1996.

[Sow84] J. F. Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[Sow87] J. F. Sowa. Semantic networks. *Encyclopedia of Artificial Intelligence*, 1987.

[Sta07] Stanford. Protégé. http ://protege.stanford.edu/, 2007.

[TH06] D. Tsarkov and I. Horrocks. Description logic reasoner : System description. In *IJCAR*, pages 292–297, 2006.

[Top01] TopicMaps.Org. Xml topic maps (xtm) 1.0. http ://www.topicmaps.org/xtm/1.0/, 2001.

[Tou] Label tourinfrance : Conditions d'échange de données entre deux systèmes d'information touristique (s.i.t.). http ://www.tourinfrance.net/tourinfrance22/index.htm.

[W3C07] W3C. Sparql query language for rdf. W3C Candidate Recommendation at http ://www.w3.org/TR/rdf-sparql-query, 2007.

Annexe A

A.1 Format TMDM

Cette partie présente en détail le modèle TMDM (cf section 2.3.1).

Le modèle *TMDM* comporte 9 classes structurées en une hiérarchie permettant l'héritage d'attributs. La figure 2.7 présente ces classes. La classe au sommet se nomme `TopicMapConstruct` et représente la super-classe de tous les constructeurs du modèle TMDM. Sa sous-classe directe est la classe `Reifiable` qui représente la super-classe des constructeurs qui sont réifiables par au plus une instance de la classe `Topic`, c'est à dire un topic au sein d'une TM. Les 6 constructeurs réifiables sont `AssociationRole`, `TopicName`, `Association`, `Occurrence`, `Variant`, `TopicMap`. En TMDM, un document *XTM 2.0* est représenté par une instance de la classe `TopicMap`. La classe `TopicMap` contient en tant qu'attributs des instances des classes `Topic` et `Association`.

Soit une instance i d'une de ces classes, pour désigner la valeur d'un de ses attributs (prenons par exemple l'attribut a) on écrit $i.[a]$. Par exemple, soit t une instance de la classe *Topic*, pour accéder à l'ensemble des valeurs de *itemIdentifiers* on écrit $t.[itemIdentifiers]$. Toute instance i d'un constructeur réifiable peut comporter au plus une instance de *Reifiable* qui si différent de *null* est le topic réifiant i (accessible par $i.[reifier]$).

Une instance tm de *TopicMap* peut contenir des instances de *T*opic et *A*ssociation (accessibles par $tm.[topics]$ ou $tm.[associations]$). Par exemple, l'instance tm de *TopicMap* associée à tm_{XTM} est telle que $acq1$, $acq2$, acq, ci, cap, web, eng, fr, tn et $tm.[associations]$ contienne a_1, a_2, a_3 avec t_i instance de *Topic* et a_j de *Association*. t_1 et t_2 correspondent aux topics $t_{1_{XTM}}$ et $t_{2_{XTM}}$ et a_1 à l'association a_{XTM}, les autres topics et les associations a_2, a_3 sont rajoutés au cours de l'interprétation TMDM du document XTM. Les associations a_2, a_3 sont de type *classe-instance* (représenté par *ci*) et permettent d'attribuer à t_1 et t_2 leur classe de topics *comp* (qui correspond à *Company*). $acq1$ à $acq2$ représentent des types de rôles et acq le type de l'association a_1. *cap*,

web,*eng*, fr et tn représentent respectivement les types d'occurrences *Capital*, *WebSite*, les domaines de validité des variations de nom du topic $t_{2_{XTM}}$ et tn le type de nom par défaut : $TopicName$. Tout x directement instance d'une classe de constructeur exceptés *TopicMap*, *Reifiable* et *TopicMapConstruct* possède un attribut *parent* dont la valeur est l'instance contenant x. Par exemple, comme l'instance tm de *TopicMap* contient une instance t_1 de *Topic*, on a $t_1.[parent] = tm$. Une instance t de la classe *Topic* (présentée à la figure A.1) peut contenir des instances de *TopicName*, *Occurrence* et *AssociationRole* (accessibles par $tm.[topicNames]$, $tm.[occurrences]$ et $tm.[roles]$). Il peut aussi comporter une instance de *Reifiable* qui si différent de *null* représente le constructeur réifié par ce topic (accessible par $t.[reified]$). t possède aussi les deux attributs *subjectLocator* et *subjectIdentifier* qui sont des ensembles de chaînes de caractères. L'instance t_1 de *Topic* dans $tm.[topics]$ est telle que $t_1.[topicNames] = \{n_{11}\}$, $t_1.[occurrences] = \{o_{11}, o_{12}\}$ et $t_1.[subjectIdentifiers]$ contient le libellé de l'élément `subjectIdentifier` dans $t_{1_{XTM}}$.

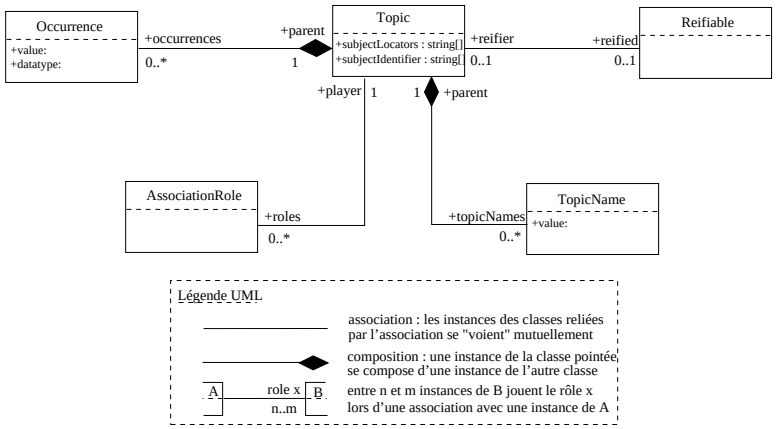


FIG. A.1 – La définition de la classe *Topic*

Pour attribuer à une instance t de *Topic* une *classe de topic*, il faut rajouter une nouvelle association a dans $tm.[association]$ et quatre instances de *Topic* t_a, t_{r1}, t_{r2} et t_c représentant respectivement le type d'association *classe-instance*, les types de rôle *classe* et *instance* et la classe du topic t. L'association a doit être telle que $a.[type]$ contienne t_a et $a.[roles]$ contienne $r1$ et $r2$; eux mêmes définis tels que $r1.[type]$ (resp. $r2.[type]$) contienne t_{r1} (resp. t_{r2}) et $r1.[player]$ (resp. $r2.[player]$) contienne t_c (resp. t). Par exemple, $t_{1_{XTM}}$ et $t_{2_{XTM}}$ possèdent un élément `instanceOf` qui établit une référence à leur classe. Cet élément est interprété en TMDM en créant dans $tm.[topics]$ les instances $inst$, $class$, ci qui représentent respectivement les types de rôle *classe* et *instance* et le type d'association *classe-instance*. Les associations a_2 et a_3 sont ajoutées à $tm.[associations]$

telles que chacune d'entre elles relie t_1 et t_2 au topic *comp* représentant leur classe.

Une instance x de *TopicName*, de *Occurrence* ou de *Association* peut posséder des instances de `Topic` dont une obligatoire à considérer comme le type de x au sein de la topic map (accessible par $x.[type]$) ; le reste étant facultatif représente le *scope* de ce constructeur (accessible par $x.[scope]$). L'occurrence o_{11} de t_1 est telle que $o_{11}.[type]$ contienne le topic *cap* représentant le type de l'occurrence (le capital d'une société). Les topics $t_{1_{XTM}}$ et $t_{2_{XTM}}$ ne présentent que des noms sans type. L'instance n associée à un nom sans type possède dans $n.[type]$ un topic qui représente le *type de nom de topic*, à savoir ici tn. Un domaine de validité est défini pour chacun des noms du topic $t_{2_{XTM}}$. Si on considère celui dont le domaine de validité est la langue française (désignons le par $n21$), alors $n21.[scope]$ contient le topic fr représentant ce domaine de validité.

Une instance n de *TopicName* peut aussi comporter des instances de `Variant` (accessible par $n.[variants]$). n possède un attribut *value* qui encode le nom principal attribué au topic. Par exemple, si on considère $t_{1_{XTM}}$, $n11.[value]$ contient le libellé de `value` dans l'élément `name`. La variation du nom $n21$ du topic $t_{2_{XTM}}$, est représentée par un objet $n21v$ instance de *Variant* dans $n21.[variants]$.

Une instance x de *Variant* ou *Occurrence* possède les attributs *value* et *datatype*. Par exemple, $n21v.[value]$ contient le libellé de `resourceData` dans l'élément `variant`. En ce qui concerne l'occurrence $o11$, $o11.[value]$ contient la valeur "1000000" et $o11.[datatype]$ désigne le type de valeur de l'occurrence (ici un entier).

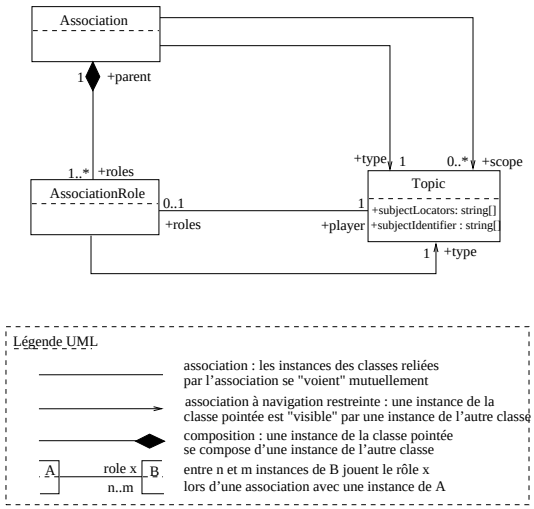


FIG. A.2 – La définition de la classe *Association*.

Une instance a de la classe *Association* présentée à la figure A.2 contient au moins

une instance r de *AssociationRole* (accessible par $a.[roles]$). Cette dernière doit posséder exactement deux instances de `Topic` l'une (accessible par $r.[type]$) à considérer comme le type de rôle et l'autre (accessible par $r.[player]$) comme le topic jouant ce rôle dans l'association auquel il appartient. Par exemple, l'association $a1$ est telle que $a1.[roles]$ contienne les rôles $r_{acq_{ing}}$ et r_{inst_1}. Si on considère $r_{acq_{ing}}$ (resp. r_{inst_1}), $r_{acq_{ing}}.[player]$ (resp. $r_{inst_1}.[player]$) contient le topic t_1 (resp. t_2) et $r_{acq_{ing}}.[type]$ (resp. $r_{inst_1}.[type]$) le topic $acq1$ (resp. $acq2$) représentant le type du rôle auquel fait référence l'élément `type`.

Le format *TMDM* est l'interprétation formelle de base d'un document *XTM 2.0* à partir de laquelle sont définis d'autres formalismes de représentation des TM. La suite en présente deux : le format des Subject Maps qui modélise les structures *TMDM* en utilisant des ensembles de couples clé-valeur et le modèle Q basé sur des ensembles de quintuplets.

A.2 Représentation d'une instance de *TMDM* en *Subject Maps*

Cette partie présente le résultat de la transformation d'une TM tm du formalisme *TMDM* dans le formalisme *SM*. D'un point de vue général, le principe consiste à associer un *SP* à chaque instance de *TMDM*. La correspondance entière entre le modèle *TMDM* et le formalisme *SM* étant assez conséquente, elle n'est pas détaillée ici. La suite présente une partie de la traduction en *SM* de l'instance *TMDM* de la classe `TopicMaps` associée au document *XTM* constituée des topics et associations présentée dans les figures 2.5 et 2.6.

L'instance tm de la classe `TopicMaps` en *TMDM* devient en *SM* le *SP* présenté à la figure A.3.

$$tm = \left\{ \begin{array}{l} (topic, lib(t_1)) \\ (topic, lib(t_2)) \\ (topic, lib(comp)) \\ (topic, lib(inst)) \\ (topic, lib(class)) \\ (topic, lib(acq_{ing})) \\ (topic, lib(acq_{ed})) \\ (topic, lib(acq)) \\ (topic, lib(c-i)) \\ (topic, lib(class)) \\ (topic, lib(cap)) \\ (topic, lib(web)) \\ (topic, lib(eng)) \\ (topic, lib(fr)) \\ (topic, lib(tn)) \\ (association, lib(a_1)) \\ (association, lib(a_2)) \\ (association, lib(a_3)) \end{array} \right\} \qquad t_1 = \left\{ \begin{array}{l} (topic\,name, lib(name_1)) \\ (occurrence, lib(occ_{web_1})) \\ (occurrence, lib(occ_{cap_1})) \\ (role\,played, lib(r_{acq_{ing}})) \\ (role\,played, lib(r_{inst_1})) \\ (subject\,identifier, lib(sub_1)) \\ (item\,identifier, lib(id_1)) \\ (parent, lib(tm)) \end{array} \right\}$$

FIG. A.3 – Deux *SP* : l'un représentant une TM et l'autre un topic.

Les éléments associés à la clé *topics* et ceux associés à la clé *associations* correspondent respectivement aux éléments de $tm.[topics]$ et de $tm.[associations]$ en TMDM.

Après sa traduction en *SM*, l'élément TMDM t_1 devient le *SP* t_1 présenté à la figure A.3. Il réunit dans les propriétés de clé *occurrences*, *topic names* et *roles played* ses noms, ses occurrences et les rôles qu'il joue dans les associations auxquelles il participe. La propriété de clé *subject identifiers* permet de rassembler les références identifiant le sujet représenté par ce topic qui se trouvent dans $t.[subjectIdentifiers]$. t_1 joue un rôle accessible dans $t_1.[rolesplayed]$ dans deux associations différentes (elles seront détaillées plus loin). Lorsqu'un *SP* contient un autre *SP* dans une de ses propriétés, le second possède une propriété de clé *parent* dont la valeur identifie le *SP* père. Par exemple, dans le cas de t_1 la valeur de cette propriété est tm puisque tm contient t_1 dans l'ensemble associé à la clé *topics*.

Le nom principal du topic t_1 de notre exemple se traduit en le *SP* $n11$ de la figure A.4.

$$name_1 = \left\{ \begin{array}{l} (value, MonitoringTechs) \\ (type, lib(tn)) \\ (scope, U) \\ (parent, lib(t_1)) \end{array} \right\} \quad occ_{cap_1} = \left\{ \begin{array}{l} (value, lib(1000000)) \\ (datatype, INT) \\ (scope, U) \\ (type, lib(cap)) \\ (parent, lib(t_1)) \end{array} \right\}$$

$$n2variant = \left\{ \begin{array}{l} (value, S.A.V.E.L.) \\ (datatype, STRING) \\ (scope, lib(fr)) \\ (parent, lib(t_2)) \end{array} \right\}$$

FIG. A.4 – Trois *SP* représentant respectivement un nom, une variation de nom et une occurrence.

Dans les exemples, les noms commençant par une majuscule désignent les *SP* correspondant à des éléments constitutifs du modèle TMDM. Comme en *XTM 2.0* les noms ont un type, tn désigne le *SP* représentant le type de nom par défaut $TopicName$. U désigne le *SP* représentant le domaine de validité le moins contraint (en général vide). Le topic t_2 comporte quant à lui deux noms principaux : l'un ayant un domaine de validité pour la langue anglaise et l'autre pour la langue française. Les *SP* $n21$ et $n22$ associées à ces noms comportent une propriété $(scope, lib(eng))$ et $(scope, lib(fr))$ avec eng et fr la traduction en *SP* des éléments TMDM représentant les contextes pour l'anglais et le français. $n21$ est présenté à la figure A.4.

Les noms du topic t_2 présentent aussi des *variations*. En *SM*, cela signifie que le *SP* associé à un des noms du topic comporte une propriété $(variants, lib(n21variant))$ (le *SP* associé à l'autre nom possède $(variants, lib(n22variant))$). $n2variant$ est présenté à la figure A.4.

Le domaine de validité de $n21v$ est celui du nom principal dont il est la variation. Les topics t_1 et t_2 présentent des occurrences. Dans le cas de t_1, l'occurrence représentant le capital de la société devient le *SP* $o11$ présenté à la figure A.4 dont la *valeur* est un *entier*, le domaine de validité est laissé libre et le type est un topic *cap* représentant la classe des valeurs de capitaux. Les *SP* associés aux topics comportent aussi des propriétés contenant la référence au sujet représenté. Dans le cas de t_1, par exemple, $s11$ est de la forme $s11 = \{(topic, lib(t_1)), (reifier, uri1 : Entite\_MonitoringTechs), (parent, lib(t_1))\}$.

En ce qui concerne les associations, il en va presque de même que pour les topics. L'association $a1$ représentant une acquisition de société est traduite en le *SP* présenté à la figure A.5.

$$a_1 = \left\{ \begin{array}{l} (role, lib(r_{acq_{ing}})) \\ (role, lib(r_{acq_{ed}})) \\ (scope, U) \\ (type, lib(acq)) \\ (parent, lib(tm)) \end{array} \right\} \qquad r_{acq_{ing}} = \left\{ \begin{array}{l} (player, lib(t_1)) \\ (type, lib(acq_{ing})) \\ (parent, lib(a_1)) \end{array} \right\}$$

FIG. A.5 – Un *SP* représentant une association et un autre représentant un des rôles joués dans l'association.

Puisqu'elle n'a pas de domaine de validité particulier, on lui en attribue un par défaut. Le type de l'association est représenté par un *SP acq*. Il faut noter que l'obtention des relations isa_m, sub_m à partir des associations $class{-}instance$, $subclass{-}superclass$ et de la propriété *type* n'est pas encore clairement définie en *SM*. Le rôle $r_{acq_{ing}}$ de l'association $a1$ se trouve parmi les rôles joués ($roles\,played$) par le topic t_1 et se présente sous la forme du *SP* de la figure A.5.

t_1 joue aussi un rôle r_{inst_1} de type instance dans une association $a2$ représentant la relation de classe à instance qui le relie à un topic *comp* jouant le rôle de type classe (il en va de même pour t_2 dans l'association de même type $a3$).

A.3 Représentation d'une TM dans le modèle Q

Cette partie présente en première partie la signification d'un modèle Q représentant une TM et en seconde partie la transformation qui permet d'obtenir ce modèle.

A.3.1 Signification d'un modèle de Q représentant une TM

Cette partie présente la signification attribuée au modèle de Q représentant une TM. Une partie immuable I_Q du vocabulaire I contient les termes utilisés en tant que prédicats de base pour définir des quintuplets de Q. Pour un quintuplet q donné, chaque $pred(q) \in I_Q$ donne à q une signification précise qui est exposée dans ce qui suit :

- $pred(q) = ItemIdentifier$ ssi $obj(q)$ est l'identifiant de $subj(q)$.
- $pred(q) = NodeUri$ ssi $obj(q)$ est une *URI* qui identifie la ressource représentée par $subj(q)$.
- $pred(q) = ScopeMember$ ssi $subj(q)$ représente le domaine de validité auquel $obj(q)$ appartient en tant que composant.
- $pred(q) = Type$ ssi $subj(q)$ est l'instance du type représenté par $obj(q)$. Dans le cas où $pred(q) = TypeInstance$ et $obj(q) = InformationResource$, l'instance doit être considérée comme une ressource (qu'un topic du modèle Q peut éventuellement désigner comme l'entité qu'il représente).
- $pred(q) = MetaType$ ssi $subj(q)$ est l'instance du type représenté par $obj(q)$ à un niveau méta. $subj(q)$ est lui même destiné à être utilisé comme type dans un autre

quintuplet. C'est pourquoi le statut de type ne lui est accordé qu'au niveau méta. Dans le cas où $obj(q)$ vaut $TopicName$ ou $Occurrence$, $subj(q)$ représente le type d'un nom de topic ou celui d'une occurrence.

- $pred(q) = AssociationType$ ssi $subj(q)$ représente l'instance du type d'association $obj(q)$ dont les instances ne mettent que deux topics en relation. Pour un tel quintuplet, il existe deux quintuplets q_o et q_s tels que $pred(q_o) = ObjectRole$, $pred(q_s) = SubjectRole$ et $subj(q_o) = subj(q_s) = subj(q)$. q_o (resp. q_s) signifie que le topic jouant le rôle $obj(q_o)$ (resp. $obj(q_s)$) dans une association de type $subj(q)$ doit être considéré comme l'*objet* (resp. le *sujet*) de la mise en relation.
- $pred(q)$ vaut $Topic$ ou $Association$ ssi $subj(q)$ représente une TM et $obj(q)$ un topic ou une association appartenant à cette TM.
- $pred(q)$ (resp. $id(q)$) est le sujet (resp. l'objet) d'un quintuplet dont le prédicat est $AssociationType$ (resp. $Association$) ssi $id(q)$ représente une association qui met en relation un topic sujet $subj(q)$ et un topic objet $obj(q)$.
- $subj(q)$ est le sujet (resp. l'objet) d'un autre quintuplet dont le prédicat est $Type$ (resp. $Association$) ssi $subj(q)$ représente une association qui met en relation $n \neq 2$ topics et dans laquelle $obj(q)$ représente un des topics parmi les n mis en relation, $id(q)$ identifie le rôle de type $pred(q)$ que joue le topic dans l'association. Il existe n quintuplets dont le sujet est $subj(q)$.
- $pred(q) = Reifies$ ssi $subj(q)$ désigne un topic qui représente un élément identifié par $obj(q)$. Le libellé $obj(q)$ doit appartenir à un autre quintuplet. On dit que le topic *réifie* l'élément.
- $pred(q)$ est le sujet d'un quintuplet dont le prédicat est $MetaType$ et l'objet est $Occurrence$ (resp. $TopicName$) ssi $id(q)$ identifie l'occurrence (resp. le nom) de type $pred(q)$ du topic $subj(q)$ et dont la valeur (resp. le libellé) est $obj(q)$.
- $pred(q) = Variant$ ssi $subj(q)$ représente un nom de topic et intervient donc en tant que *sujet* dans un quintuplet de *prédicat* $MetaType$ et d'*objet* $TopicName$. $id(q)$ identifie le quintuplet et $obj(q)$ sa valeur (qui est en général le libellé désigné comme étant la variation du nom du topic).
- lorsque le contexte d'un quintuplet q n'est pas spécifié, par défaut il prend la valeur $S_Q \in I_Q$. $con(q) = S_Q$ signifie que le quintuplet représente une primitive de construction interne et non pas des connaissances. Lorsque $pred(q)$ est le sujet d'un quintuplet dont le prédicat est $AssociationType$, $MetaType$, $Type$ ou $Variant$ alors $con(q)$ désigne le domaine de validité de q et prend par défaut la valeur $S_U \in I_Q$ qui représente le contexte vide (indéfini). Si $con(q) \neq S_U$, il apparaît en tant que *sujet* dans un quintuplet de prédicat $ScopeMember$.
- lorsque l'identifiant d'un quintuplet q n'est pas spécifié, il prend la valeur par défaut $\varnothing$. Cela signifie que q n'est pas identifiable (il ne peut donc pas intervenir dans un autre quintuplet en tant qu'objet ou sujet puisque pour y parvenir il faudrait l'identifier).

A.3.2 Représentation d'une instance *TMDM* sous la forme d'un modèle de Q

Les exemples présentés dans cette partie correspondent à la représentation sous la forme d'un modèle de Q de l'objet tm_{TMDM} décrit en *TMDM* en section A.1. Pour les besoins de l'interprétation, on se donne deux fonctions $i(x)$ et $si(x)$. La première attribue un identifiant unique de A à un élément *TMDM* et lorsqu'on écrit $i(t, x)$ elle permet d'interpréter la valeur représentée par le libellé x selon le type de donnée t spécifié. La seconde attribue un identifiant à un ensemble fini X de topics donné telle que $si(\emptyset) = S_U$. Je précise que la fonction i n'est pas utilisée dans [Gar05a]. Elle a été introduite ici car il semblait préférable de distinguer les éléments sources en *TMDM* des éléments cibles du modèle de Q. Soit x un objet *TMDM* et q un quintuplet de M, ces deux objets entretenant une correspondance unique, on désigne par M le modèle de Q correspondant à un l'objet *TopicMap tm* en *TMDM*.

- $pred(q) = ItemIdentifier$ ssi $subj(q) = i(x)$ et $obj(q) \in x.[itemIdentifier]$.
- $pred(q) = NodeURI$ ssi $subj(q) = i(x)$ et $obj(q) \in x.[subjectIdentifiers] \cup x.[subjectLocators]$ et x instance de *Topic*. Par exemple, un tel quintuplet avec $subj(q) = i(t_1)$ et $obj(q) =$ "$uri : Entite\_MonitoringTechs$" correspond à l'objet *Topic* t_1 dans $tm_{TMDM}.[topics]$.
- $pred(q) = ScopeMember$ ssi $subj(q) = si(s)$ et $obj(q) = i(t)$ tels que $s = x.[scope]$ et $t \in s$ instance de *Topic*. Dans l'exemple, $obj(q) = i(eng)$, $subj(q) = si(s)$ avec $s = \{eng\} = n.[scope]$ et n est un nom du topic t_2.
- $pred(q) = Type$ ssi $subj(q) = i(x)$, $obj(q) = i(x.[type])$, $con(q) = si(x.[scope])$, x est instance d' *Association* et le nombre d'éléments de $x.[roles]$ est différent de deux. L'exemple présente une association reliant deux topics ce qui ne correspond pas à ce cas.
- $pred(q) = TypeInstance$ et $obj(q) = InformationResource$ ssi $subj(q) = i(x)$, $x.[subjectLocators]$ est vide et x est instance de *Topic*.
- $pred(q) = MetaType$ et $obj(q)$ a pour valeur $TopicName$ (resp. $Occurrence$) ssi $subj(q) = i(x.[type])$ et x est instance de *TopicName* (resp. *Occurrence*). Dans l'exemple, un quintuplet de cette forme tel que $subj(q) = i(tn)$ correspond au type tn du nom n du topic t_1.
- $pred(q) = AssociationType$ et il existe q_o et q_s dans M tels que $pred(q_o) = ObjectRole$, $pred(q_s) = SubjectRole$ et $subj(q_o) = subj(q_s) = subj(q)$ ssi $obj(q) = i(x.[type])$, $x.[roles] = \{r_1, r_2\}$, x est instance d'$Association$, $obj(q_s) = i(r_1.[type])$ et $obj(q_o) = i(r_2.[type])$. L'exemple présente un type d'association $acq \in tm_{TMDM}.[topics]$ dont les instances ne mettent en relation que deux topics jouant des rôles de types $acq1$ et $acq2$. Les quintuplets q, q_o et q_s correspondant à cette association vérifient que $subj(q) = t_{acq}$, $obj(q) = i(acq)$, $obj(q_s) = i(acq1)$ et $obj(q_o) = i(acq2)$
- $pred(q)$ (resp. $id(q)$) est le sujet (resp. l'objet) d'un quintuplet dont le prédicat

est *AssociationType* (resp. *Association*) ssi $id(q) = i(x)$, $x.[roles] = \{r_1, r_2\}$, $subj(q) = r_1.[player]$ $obj(q) = r_2.[player]$ et x est instance d'*Association*. Si on se réfère à l'exemple, un quintuplet de cette forme est associé à l'association a de type *acq*.

- $subj(q)$ est le sujet (resp. l'objet) d'un autre quintuplet dont le prédicat est *Type* (resp. *Association*) ssi $subj(q) = i(x)$, r appartient à $x.[roles]$ dont le nombre d'éléments n'est pas deux, $pred(q) = i(r.[type])$, $id(q) = i(r)$, $con(q) = si(x.[scope])$ et $obj(q) = i(r.[player])$ avec x une instance d'*Association*. L'exemple ne présente que des associations binaires qui ne correspondent pas à ce cas de figures.
- $pred(q)$ a pour valeur *Topic* (resp. *Association*) ssi $subj(q) = i(x)$ et $obj(q)$ a pour valeur $i(t)$ (resp. $i(a)$) avec $t \in x.[topics]$ instance de *Topic* (resp. $a \in x.[associations]$ instance d' *Association*) et x instance de *TopicMap*. Si on considère l'exemple, un tel quintuplet est associé au topic t_1 tel que $subj(q) = i(tm_{TMDM})$ et $obj(q) = i(t_1)$.
- $pred(q) = Reifies$ ssi $subj(q) = i(x)$, $obj(q) = i(x.[reified])$ et x est instance de *Topic*.
- $pred(q)$ est le sujet d'un autre quintuplet dont le prédicat est $MetaType$ et l'objet est $Occurrence$ (resp. $TopicName$) ssi $y \in x.[occurrences]$ (resp. $y \in x.[topicNames]$) , $subj(q) = i(x)$, $pred(q) = i(y.[type])$, $id(q) = i(y)$, $con(q) = si(y.[scope])$ et $obj(q) = i(y.[datatype], y.[value])$ (resp. $obj(q) = y.[value]$) sachant que x est instance de *Topic*. Par exemple, un tel quintuplet correspond à l'occurrence o_{12} du topic t_1 tel que $subj(q) = i(t_1)$, $pred(q) = i(web)$, $id(q) = i(o_{12})$, $con(q) = S_U$ et $obj(q) = i(URI, "http://www.monitoringtechs.com")$.
- $pred(q) = Variant$ ssi $y \in x.[variants]$, $subj(q) = i(x)$, $id(q) = i(y)$, $con(q) = si(y.[scope])$ et $obj(y) = i(y.[datatype], y.[value])$ sachant que x est instance de *TopicName*.

A.4 Première transformation - représentation des règles

Le quatrième chapitre présente trois types de transformations entre les TM et la famille $\mathcal{SG}$. La première transformation permet de traiter en toute généralité les connaissances des trois niveaux d'ITM sans se soucier du niveau sur lequel elle est appliquée. Cette transformation produit des graphes très volumineux au regard de la taille des TM sources. Elle représente aussi les propriétés sémantiques des niveaux méta et modèles sous la forme de règles ou de contraintes de la famille $\mathcal{SG}$. La figure A.6 présente la règle de transitivité de l'association de type `classe-sousclasse` qui consiste à dire que si une classe est sous-classe d'une seconde et super-classe d'une troisième alors la seconde est super-classe de la troisième. Si on considère, dans le sens des aiguilles d'une montre, chacun des quatre sous-graphes entourés de pointillés ; ils représentent respecti-

vement l'attribution à un élément de base de la classe `Classe` (sommet `[T :C]`), du type d'association `Classe-sousclasse` (sommet `[T :SC]`) et des types de rôle `super-classe` (sommet `[T :RC+]`) et `sous-classe` (sommet `[T :RC-]`). Le motif central entouré de pointillés représente la propriété de transitivité de manière générale pour des éléments associations, rôles et topics quelconques. En reliant chaque sommet du motif central avec un des motifs périphériques, on représente l'instanciation de ces éléments et on parvient à restreindre la portée de la transitivité aux sommets concepts représentant les topics (resp. associations) de classe `Classe` (resp. de type `Classe-sousclasse`) et aux sommets relations représentant les rôles de type `super-classe` ou `sous-classe`.

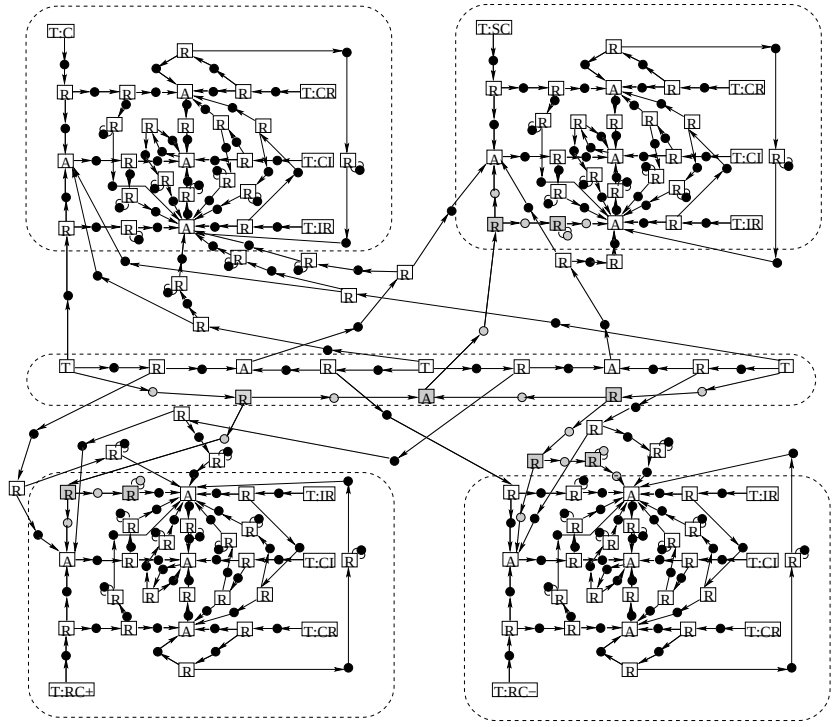


FIG. A.6 – Règle *SG* attribuant la propriété de transitivité au lien de sous-classement.

A.5 Algorithme de suppression de connaissances.

Nous avons vu en conclusion qu'une des principales perspectives de recherche concernait la suppression de connaissances valides dans ITM. Ce type de suppression doit conserver *saturé* le SG correspondant à la portion valide de ITM. La suite présente l'algorithme permettant de réaliser une telle suppression. La validité de cet algorithme est

en cours de démonstration.

Soit G un SG et $G^{\mathcal{R}}$ la fermeture de G par rapport aux règles $\mathcal{R}$, l'algorithme permet de supprimer un sous-graphe H du graphe $G^{\mathcal{R}}$ tout en conservant le résultat G' saturé par rapport à $\mathcal{R}$; G' étant un sous-graphe de $G^{\mathcal{R}}$. Le principe consiste à déterminer le sous-graphe H^+ de $G^{\mathcal{R}}$ qui n'apparaît pas dans G' et à le supprimer de $G^{\mathcal{R}}$. L'algorithme fonctionne en deux étapes.

La première étape reconstitue une saturation à partir de H dans $G^{\mathcal{R}}$. Pour ce faire, on dispose d'un ensemble I qui contient au démarrage de l'algorithme les sommets de H. La reconstitution consiste à projeter entièrement chaque règle dans $G^{\mathcal{R}}$ sur au moins un sommet de I, puis à ajouter dans I les images de la conclusion par cette projection ; et ceci jusqu'à stabilisation de l'ensemble I. A la fin de l'algorithme, I contient les sommets produits par une saturation incrémentale démarrant à partir de H dans $G^{\mathcal{R}}$. Le graphe induit par I contient donc H^+ mais il peut aussi contenir d'autres sommets qu'il faut conserver : le sous-ensemble (éventuellement vide) I' de I des sommets inférés à partir d'un sous-graphe H' de $G^{\mathcal{R}}$ disjoint de H.

La seconde étape prend ce cas en considération et après avoir supprimé le sous-graphe induit par I, elle relance une saturation *incrémentale* à partir des sommets adjacents au sous-graphe supprimé (c'est à dire l'ensemble constitué des sommets voisins des sommets supprimés) ce qui permet de rajouter les éventuels sommets de I' qui manquent au graphe résultant de la suppression pour être saturé par rapport aux règles.

Une fois ces étapes achevées, l'algorithme détecte et retourne les contraintes positives violées ou réparées par la suppression. Comme le graphe de la base avant suppression respecte les contraintes négatives, il en est de même pour celui résultant de la suppression, puisque ce dernier est toujours plus général que le premier (l'opération de suppression est généralisante).

A.6 Format XTM 2.0

Cette section présente les définitions en *XTM 2.0* des topics $t_{1_{XTM2}}$ et $t_{2_{XTM2}}$ et de l'association a_{XTM2} présentés à la section 2.2.2. La définition en *XTM 2.0* du topic t_{1XTM2} représentant la société MonitoringTechs est de la forme :

```
<topic>
  <itemIdentity>"t1_XTM2"</itemIdentity>
  <instanceOf>
    <topicRef xlink:href="uri1:Company" />
  </instanceOf>
  <subjectIdentifier xlink:href=
    "uri1:Entité\_MonitoringTechs" />
  <name>
    <value>MonitoringTechs</value>
  </name>
```

```
  <occurrence>
    <type>
      <topicRef xlink:href="uri1:Capital" />
    </type>
    <resourceData datatype="integer">
      1000000</resourceData>
  </occurrence>
  <occurrence>
    <type>
      <topicRef xlink:href="uri1:WebSite" />
    </type>
    <resourceData datatype="string">
      http://www.monitoringtechs.com
    </resourceData>
  </occurrence>
</topic>
```

La définition en *XTM 2.0* du topic t_{2XTM} représentant la Société d'analyse et de veille économique du Languedoc est :

```
<topic>
  <itemIdentity>"t2_XTM2"</itemIdentity>
  <instanceOf>
    <topicRef xlink:href="uri1:Company" />
  </instanceOf>
  <subjectIdentifier xlink:href=
    "uri1:Entité\_SAVEL" />
  <name>
    <scope>
      <topicRef xlink:href="uri1:Français" />
    </scope>
    <value> Société d'analyse et de veille
        économique du Languedoc </value>
    <variant>
        <resourceData>S.A.V.E.L.</resourceData>
    </variant>
  </name>
  <name>
    <scope>
      <topicRef xlink:href="uri1:English" />
    </scope>
    <value> Languedoc Economic Analyse
        And Monitoring Company </value>
    <variant>
        <resourceData>L.E.A.M.C</resourceData>
    </variant>
  </name>
  <occurrence>
    <type>
```

```
      <topicRef xlink:href="uri1:Capital" />
    </type>
  <resourceData datatype="integer">
    50000</resourceData>
  </occurrence>
</topic>
```

Ci-suit la définition de l'association a_{XTM2} en *XTM 2.0*.

```
<association id="a_XTM">
  <type>
    <topicRef xlink:href="uri1:Acquisition" />
  </type>
  <role>
    <type>
        <topicRef xlink:href="uri1:Acquiring" />
    </type>
    <topicRef xlink:href="t1_XTM2" />
  </role>
  <role>
    <type>
        <topicRef xlink:href="uri1:Acquired" />
    </type>
    <topicRef xlink:href="t2_XTM2" />
  </role>
</association>
```

A.7 XTMX

Ci-suit la DTD correspondant à la version étendue par Mondeca du XTM 1.0.

```
<!ELEMENT topicMap (topic | association | mergeMap)*>
<!ATTLIST topicMap
id ID #IMPLIED
xmlns CDATA #FIXED "http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
xml:base CDATA #IMPLIED
>

<!ELEMENT topic (instanceOf*, mondecaMetadata?, subjectIdentity?, (baseName | occurrence)*)>
<!ATTLIST topic
id ID #REQUIRED
>

<!ELEMENT instanceOf (topicRef | subjectIndicatorRef)>
<!ATTLIST instanceOf
id ID #IMPLIED
```

```
>

<!ELEMENT mondecaMetadata (mondecaQuality?, mondecaSource?, mondecaSourceDocument?, mondecaPosition?,
  mondecaTrust?)>
<!ATTLIST mondecaMetadata
id ID #IMPLIED
beginDate CDATA #IMPLIED
endDate CDATA #IMPLIED
>

<!ELEMENT mondecaQuality (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaQuality
id ID #IMPLIED
>

<!ELEMENT mondecaSource (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaSource
id ID #IMPLIED
>

<!ELEMENT mondecaTrust (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaTrust
id ID #IMPLIED
>

<!ELEMENT mondecaSourceDocument EMPTY>
<!ATTLIST mondecaSourceDocument
id ID #IMPLIED
xlink:href CDATA #REQUIRED
>

<!ELEMENT mondecaPosition EMPTY>
<!ATTLIST mondecaPosition
id ID #IMPLIED
start CDATA #IMPLIED
length CDATA #IMPLIED
>

<!ELEMENT subjectIdentity (resourceRef?, (topicRef | subjectIndicatorRef)*)>
<!ATTLIST subjectIdentity
id ID #IMPLIED
>

<!ELEMENT topicRef EMPTY>
<!ATTLIST topicRef
id ID #IMPLIED
xlink:type NMTOKEN #FIXED "simple"
xlink:href CDATA #REQUIRED
>
```

```
<!ELEMENT subjectIndicatorRef EMPTY>
<!ATTLIST subjectIndicatorRef
id ID #IMPLIED
xlink:type NMTOKEN #FIXED "simple"
xlink:href CDATA #REQUIRED
>

<!ELEMENT baseName (scope?, mondecaMetadata?, mondecaDataItemType?, baseNameString, variant*)>
<!ATTLIST baseName
id ID #IMPLIED
>

<!ELEMENT mondecaDataItemType (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaDataItemType
id ID #IMPLIED
>

<!ELEMENT baseNameString (#PCDATA)>
<!ATTLIST baseNameString
id ID #IMPLIED
>

<!ELEMENT variant (parameters, variantName?, variant*)>
<!ATTLIST variant
id ID #IMPLIED
>

<!ELEMENT variantName (resourceRef | resourceData)>
<!ATTLIST variantName
id ID #IMPLIED
>

<!ELEMENT parameters (topicRef | subjectIndicatorRef)+>
<!ATTLIST parameters
id ID #IMPLIED
>

<!ELEMENT occurrence (instanceOf?, scope?, mondecaMetadata?, mondecaDataItemLinkTitle?,
  (resourceRef | resourceData | mondecaDataItemPointer | mondecaDataItemParameter |
  mondecaDataItemNumeric | mondecaDataItemTime))>
<!ATTLIST occurrence
id ID #IMPLIED
>

<!ELEMENT mondecaDataItemLinkTitle (resourceData)>
<!ATTLIST mondecaDataItemLinkTitle
id ID #IMPLIED
>
```

```
<!ELEMENT resourceRef EMPTY>
<!ATTLIST resourceRef
id ID #IMPLIED
xlink:type NMTOKEN #FIXED "simple"
xlink:href CDATA #REQUIRED
>

<!ELEMENT resourceData (#PCDATA)>
<!ATTLIST resourceData
id ID #IMPLIED
>

<!ELEMENT association (instanceOf?, scope?, mondecaMetadata?,
  (mondecaBaseName | mondecaOccurrence)*, member+)>
<!ATTLIST association
id ID #IMPLIED
>

<!ELEMENT mondecaBaseName (scope?, mondecaMetadata?, mondecaDataItemType?, baseNameString, variant*)>
<!ATTLIST mondecaBaseName
id ID #IMPLIED
>

<!ELEMENT mondecaOccurrence (instanceOf?, scope?, mondecaMetadata?, mondecaDataItemLinkTitle?,
  (resourceRef | resourceData | mondecaDataItemPointer | mondecaDataItemParameter |
   mondecaDataItemNumeric | mondecaDataItemTime))>
<!ATTLIST mondecaOccurrence
id ID #IMPLIED
>

<!ELEMENT mondecaDataItemPointer (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaDataItemPointer
id ID #IMPLIED
weight NMTOKEN #IMPLIED
>

<!ELEMENT mondecaDataItemParameter (mondecaReferenceTable?, mondecaDefaultValue?)>
<!ATTLIST mondecaDataItemParameter
id ID #IMPLIED
cardinality NMTOKEN #IMPLIED
order NMTOKEN #IMPLIED
orientation NMTOKEN #IMPLIED
>

<!ELEMENT mondecaReferenceTable (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaReferenceTable
id ID #IMPLIED
>
```

```
<!ELEMENT mondecaDefaultValue (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaDefaultValue
id ID #IMPLIED
>

<!ELEMENT mondecaDataItemNumeric (mondecaNumericValue, mondecaNumericUnit?)>
<!ATTLIST mondecaDataItemNumeric
id ID #IMPLIED
>

<!ELEMENT mondecaNumericValue (#PCDATA)>
<!ATTLIST mondecaNumericValue
id ID #IMPLIED
>

<!ELEMENT mondecaNumericUnit (topicRef | subjectIndicatorRef)>
<!ATTLIST mondecaNumericUnit
id ID #IMPLIED
>

<!ELEMENT mondecaDataItemTime (mondecaTimeBegin?, mondecaTimeEnd?)>
<!ATTLIST mondecaDataItemTime
id ID #IMPLIED
time NMTOKEN #IMPLIED
year NMTOKEN #IMPLIED
month NMTOKEN #IMPLIED
date NMTOKEN #IMPLIED
hour NMTOKEN #IMPLIED
minute NMTOKEN #IMPLIED
second NMTOKEN #IMPLIED
>

<!ELEMENT mondecaTimeBegin EMPTY>
<!ATTLIST mondecaTimeBegin
id ID #IMPLIED
time CDATA #IMPLIED
year NMTOKEN #IMPLIED
month NMTOKEN #IMPLIED
date NMTOKEN #IMPLIED
hour NMTOKEN #IMPLIED
minute NMTOKEN #IMPLIED
second NMTOKEN #IMPLIED
>

<!ELEMENT mondecaTimeEnd EMPTY>
<!ATTLIST mondecaTimeEnd
id ID #IMPLIED
time CDATA #IMPLIED
```

```
year NMTOKEN #IMPLIED
month NMTOKEN #IMPLIED
date NMTOKEN #IMPLIED
hour NMTOKEN #IMPLIED
minute NMTOKEN #IMPLIED
second NMTOKEN #IMPLIED
>

<!ELEMENT member (roleSpec?, mondecaMetadata?, (mondecaBaseName | mondecaOccurrence |
  topicRef | resourceRef | subjectIndicatorRef)*)>
<!ATTLIST member
id ID #IMPLIED
>

<!ELEMENT roleSpec (topicRef | subjectIndicatorRef)>
<!ATTLIST roleSpec
id ID #IMPLIED
>

<!ELEMENT scope (topicRef | resourceRef | subjectIndicatorRef)+>
<!ATTLIST scope
id ID #IMPLIED
>

<!ELEMENT mergeMap (topicRef | resourceRef | subjectIndicatorRef)*>
<!ATTLIST mergeMap
id ID #IMPLIED
xlink:type NMTOKEN #FIXED "simple"
xlink:href CDATA #REQUIRED
>
```