Gestion et découverte de compétences dans des environnements hétérogènes

Dong Cheng

Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Gestion et découverte de compétences dans des environnements hétérogènes

THÈSE

présentée et soutenue publiquement le 9 octobre 2008

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

CHENG Dong

Composition du jury

Président :	Michaël Petit, Professeur, Université de Namur (B)
Rapporteurs :	Giuseppe Berio, Chercheur, Università degli Studi di Torino (I) Flavio Oquendo, Professeur, Université de Bretagne-Sud (F)
Examinateurs :	Nacer Boudjlida, Professeur, Nancy Université, UHP Nancy 1, LORIA (F) Michaël Petit, Professeur, Université de Namur (B) Michaël Rusinowitch, Directeur de Recherche, INRIA Lorraine (F) François Charoy, Maître de conférences, HDR, Nancy Université, ESIAL, LORIA (F)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Remerciements

Je tiens à remercier en tout premier lieu Monsieur Nacer Boudjlida, professeur à l'université Henri Poincaré Nancy 1, qui a dirigé cette thèse depuis mon stage de D.E.A. Tout au long de ces cinq années, il a toujours été disponible pour d'intenses et rationnelles discussions. Pour tout cela, sa confiance et son soutien financier, je le remercie vivement.

Je remercie les rapporteurs de cette thèse, Monsieur Giuseppe Berio, " chercheur confirmé " à l'université de Turin, et Monsieur Flavio Oquendo, profeseur à l'université de Bretagne Sud, pour la rapidité avec laquelle ils ont lu mon manuscrit et l'intérêt qu'ils ont porté à mon travail.

Je remercie Michaël Petit, professeur à l'université de Namur, pour avoir accepté d'être président du jury et s'être intéressée à ma thèse. Merci également aux autres membres du jury qui ont accepté de juger ce travail : Monsieur Michaël Rusinowitch, Directeur de recherche à l'INRIA Grand Est, et Monsieur François Charoy, HDR, maître de conférences à l'ESIAL, Université Henri Poincaré Nancy 1.

Je remercie tous les membres de l'équipe ECOO du LORIA pour l'amitié, leur bonne humeur et le soutien qu'ils m'ont témoignés. Je pense particulièrement à Claude Godart, Pascal Molli, Khalid Benali, Hala Skaf, François Charoy, Olivier Perrin, Pascal Urso.

Je remercie tous mes amis, je pense surtout à Adnene, Calvin, Rick, WeiMing, Joseph, JingSan, Bing, Jian, Ustun et toute la troupe du très convivial Laboratoire LORIA.

Finalement j'adresse un grand merci à toute ma famille qui a toujours été présente lorsque j'en ai eu besoin.

Table des matières

Résumé

Dans le cadre de cette thèse, nous nous intéressons au problème général de la gestion et de la découverte de compétences dans des environnements hétérogènes. Nous avons étendu un langage de Logique de Description pour permettre la description et la gestion de compétences. Nos principaux apports sont :
1. définition d'un langage de la description de compétences $\mathcal{ALN}_{r+}$, fondé sur les logiques de description, pour supporter de la gestion de compétence et les services d'inférence associés ;
2. définition d'un algorithme de recherche de compétences dont la caractéristique est, en cas d'échec dans la recherche d'une réponse " exacte ", d'essayer de déterminer des réponses composées, i.e. de trouver des individus dont la complémentarité des compétences satisfait la recherche ;
3. conception d'un système de médiation fédéré pour la validation expérimentale de nos propositions. Ce système a la particularité de prendre en compte les situations où les représentations des compétences sont distribuées et homogènes (i.e. les descriptions des capacités sont exprimés dans un langage de description unique) aussi bien qu'hétérogènes (i.e. les descriptions des capacités sont exprimés dans des langages de description différents).

Mots-clés : Médiation, Hétérogénéité, Réponse composite, Description Logiques, Découverte de compétences

Abstract

This thesis addresses the general problem of capability management and capability discovery in heterogeneous environments. We increased a Description Logics language to support capability description, management and discovery. Our main contributions are :
1. the definition of a capability description language $\mathcal{ALN}$, based on Description Logic, to support capability management and the associated inference services ;
2. The definition of a capability discovery algorithm whose characteristics is, in case of failure when seeking for an "exact" answer, to try to find a composite answer, i.e. to find several individuals whose capabilities complementarily satisfy the capability which is looked for ;
3. The design and the partial implementation of a federated mediation system that serves as a platform for the experimental validation of our proposals. A distinguishing feature of this system is that it deals with the situations where capabilities descriptions are distributed among the federated mediators and where they are homogeneous (i.e. they are expressed in the same capability description language) as well as heterogeneous (i.e. they are expressed in different capability description languages).

Keywords : Mediation, Heterogeneous, Composite Answer, Description Logics, Capability Discovery

Résumé de la thèse en français

La motivation essentielle de ce travail est d'apporter une contribution à la satisfaction de besoins en recherche d'individus (que nous appellerons aussi entités) qui peuvent réaliser des actions, ainsi que la recherche d'individus qui peuvent, de façon complémentaire, réaliser des actions.

Dans la section 1, nous introduisons une définition du terme "capability" (traduit en français par capacité) et nous le relions aux notions de compétence et de connaissance. Nous introduisons également, de façon intuitive, la notion de complémentarité de compétences avant de mettre en évidence les concepts et les mécanismes requis pour atteindre nos objectifs. Le paragraphe 2 résume nos objectifs ainsi que les contributions attendues alors que le paragraphe 3 détaille la structure et le contenu de ce document.

1 Chapitre 1 : Exposé du problème : Capacités, Compétences et Connaissances

Dans ce qui suit, nous appellerons capacité la possibilité de réaliser des actions. Cette définition est clairement très proche de celle de compétence et elle n'est pas indépendante de celle de connaissance. En effet, dans ces définitions informelles, il y a une légère distinction entre ce qu'est une capacité et ce qu'est une compétence telle que définie dans le domaine des ressources humaines : "état ou qualité d'être bien qualifié pour réaliser un rôle spécifique" (*"the state or quality of being adequately or well qualified, having the ability to perform a specific role"*) [86]). De ce fait, dans ce document, nous utiliserons le terme capacité (*capability*) dans le sens defini ci-avant et nous éviterons d'utliser le terme compétence vu que ce dernier est très souvent associé au domaine de la gestion des ressources humaines. Cette utilisation du terme capacité est justifiée par le fait que nous estimons que nos propositions peuvent trouver leur application dans de multiples domaines comme la gestion de compétences, la composition de services Web, la gestion de connaissances d'entreprises, les affaires électroniques (*e-business*), etc.

Des concepts relatifs à la gestion des compétences sont définis et comparés dans la hiérarchie de la figure 1 [103]. Dans cette hiérachie, capacité et compétence sont très liés : *"Capabilities are repeatable patterns of action in the use of assets to create, produce and/or offer products to a market. Because capabilities are intangible assets that determine the uses of tangible assets and other kinds of intangible assets, capabilities are considered to be an important special category of assets. Capabilities arise from the coordinated activities of groups of people who pool their individual skills in using assets to generate organizational action."*

Dans [46], la gestion de compétences consiste en la définition, l'utilisation optimale et le développement de compétences d'enterprises, de groupes et d'individus et un système de gestion de compétences doit supporter quatre fonctions : *(i)* l'identification des compétences, *(ii)* leur évaluation, *(iii)* leur acquisition et *(iv)* leur utilisation. L'identification consiste à mettre les compétences sous une représentation formelle qui puisse être exploitée dans l'évaluation et l'utilisation des compétences. En outre, l'évaluation consiste à établir une relation entre des individus et des compétences.

De façon similaire, dans le cadre des systèmes de gestion de connaissances d'entreprise,

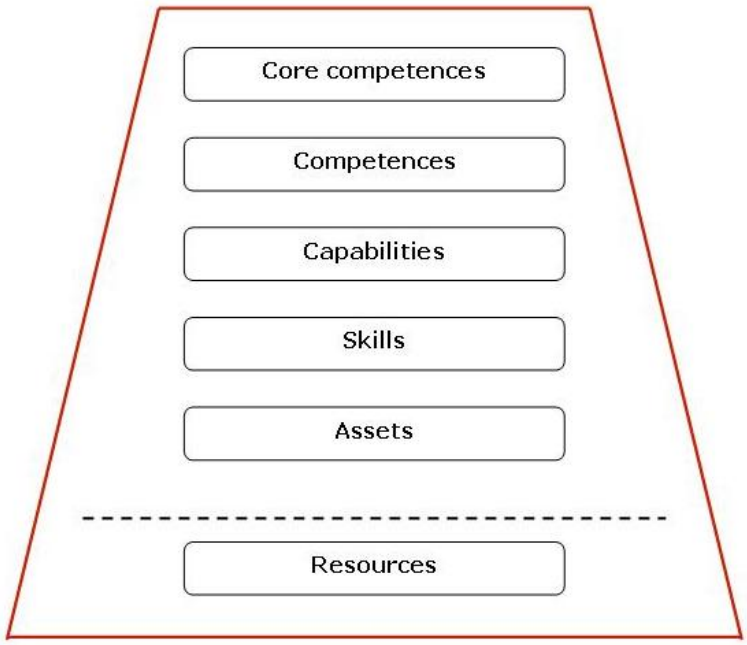


FIG. 1 – Hiérarchie des valeurs (*Assets*) [103]

[75] identifie différentes facettes de la connaissance d'entreprises, tel que schématisé sur la figure 2.

La connaissance d'entreprises (*explicit knowledge*) comme son savoir-faire (*tacit knowledge*) sont essentiels pour les processus de décision ainsi que pour l'exécution des processus constituant l'activité de l'entreprise. De ce fait, elle nécessite d'être identifiée, localisée, caractérisée, "cartographiée", évaluée et hiérarchisées pour servir les objectifs de l'entreprise.

En informatique, le terme capacité a été introduit, entre autres, par Dennis and Van Horn en 1966 dans le papier *"Programming Semantics for Multiprogrammed Computations"* [30]. L'idée de base y est la suivante : supposons qu'on conçoive un système pour accéder à un objet ; un programme doit alors avoir un jeton spécial. Ce jeton désigne un objet et donne au programme le droit de réaliser un ensemble d'actions (comme lire, écrire) sur l'objet. Un tel jeton est dénommé capacité.

Enfin, en intelligence artificielle, des capacités sont offertes par des agents intelligents et elles définissent les actions qu'un agent peut réaliser de façon isolée ou coopérative [107].

Toutes ces considérations révèlent le besoin, au minimum, d'explicitation des capacités, de leur découverte et de leur composistion. En outre, comme déjà mentionné, la découverte (dynamique) de compétences (ou services) qu'une "entité" offre a différents domaines d'application. La programmation par composants, les affaires électroniques et même la gestion des connaissances d'entreprises font partie de ces domaine d'application. De ce fait, la seule description syntaxique des services d'une entité (comme la signature d'un composant logiciel) n'est pas suffisante pour satisfaire une demande de service. Un description sémantique est requise. De même, l'explicitation des relations éventuelles entre services peut contribuer à la découverte du "meilleur" service ou des "meilleurs

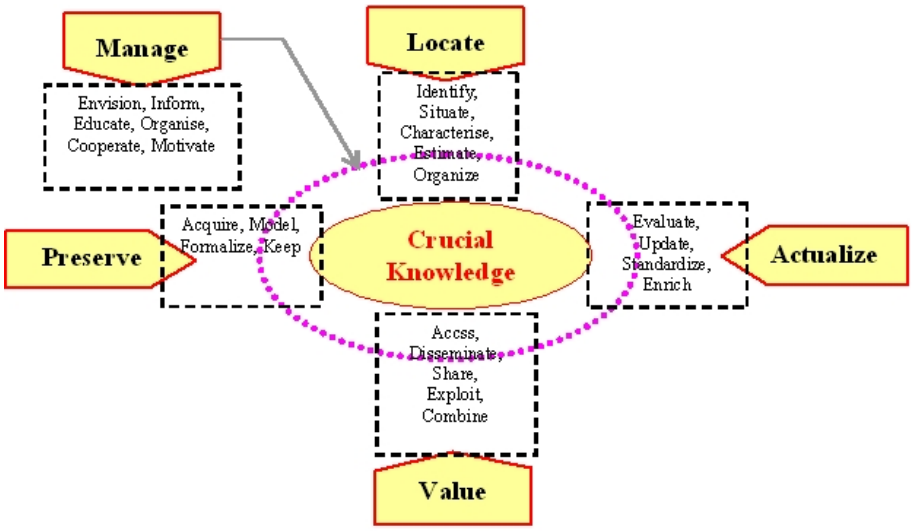


FIG. 2 – Les facettes de la gestion des connaissances d'entreprises

services complémentaires" satisfaisant une demande. Par exemple, dans le domaine du *e-business*, cette notion complémentarité (similaire à la notion de *pooling of individual skills* dans [103]) peut être appliquée lors de la tentative de constitution d'alliances ou lors de la recherche de partenaires. La découverte de la compémentarité pourra alors être réalisée si des relations entre capacités sont connues et explicitées.

De fait il y a une dépendances entre des capacités, i.e. posséder une capacité ou une combinaison de capacités a pour conséquence la possession d'autres capacités. Brian R. Gaines[39] a étendu la classification des capacités has extended the classification of capabilities as follows : *"It is tempting to extend this classification to the knowledge underlying the capabilities but this would be misleading since there is not a one-to-one relationship between knowledge and capabilities–usually, many different sets of knowledge can lead to the same capability".*

Le travail rapporté dans cette thèse tente d'apporter un cadre formel et une solution pragmatique pour l'implémentation d'un système de gestion de compétences qui permette la description, l'organisation et la découverte de compétences dans des environnements hétérogènes.

Le but et un survol de la thèse sont décrits dans la section qui suit.

2 Survol de la thèse et de ses contributions

La figure 3 donne une vision synthétique de nos travaux, montrant un volet gestion (*management*) de capacités, un volet découverte (*Discovery*) ainsi qu'un volet *architecture de médiation.*

En faisant référence aux actions induites par les diverses facettes de la gestion des connaissances d'entreprises (figure 1.2), notre travail porte plus précisément sur :

1. l'identification et l'organisation en hiérarchies de capacités possédées par une "enti-

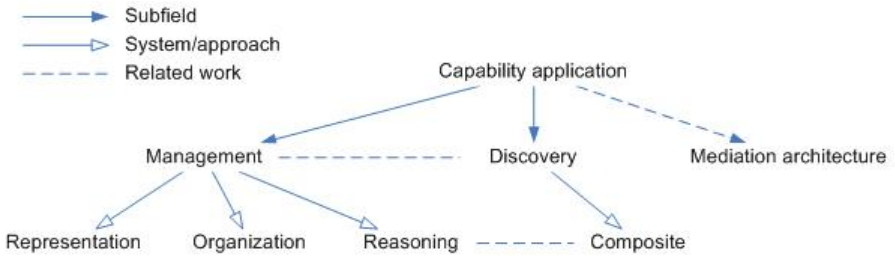


FIG. 3 – Vision synthétique d'une application de gestion de capacités

té", une entité pouvant être un individu physique ou pas,

2. la modélisation et la formalisation des capacités,
3. l'accès et la combinaison de capacités pour satisfaire un but donné, exprimé comme une requête.

Identification, organisation, modélisation and formalisation sont relatifs au volet *capability management* de notre travail et ils sont introduits dans le paragraphe **??**, alors que l'accès et la combinaison sont relatifs à la découverte de capacités (section 2.2). Finalement, la section 2.3 introduit les architectures de médiation pour supporter la recherche de capacités complémentaires entre individus.

2.1 Gestion de capacités

Nous abordons la gestion de capacités selon deux angles : la représentation de capacités et le raisonnement sur les capacités.

Dans l'angle représentation, nous nous inspirons, évidemment, de la représentation des connaissances en Intelligence Artificielle. Néanmoins, les langages de représentation des connaissances (Knowledge Representation (ou KR) Languages) ne permettent pas de représenter de façon adéquate la notion de capacités. Par exemple, le fait que la capacité "être frère de" soit transitive ou que la paire "être père de" et "être enfant de" soit symétrique n'est pas facilement exprimable dans les langages de représentation de connaissances usuels. De ce fait, nous étendons un de ces langages pour pouvoir représenter des capacités et leurs caractéristiques. Le but de la représentation est évidemment à des fins de raisonnement.

En outre, d'un point de vue purement représentation des connaissances et indépendamment de tout langage de repésentation, les verbes, plutôt que les noms permettent de discriminer des documents par types et par des propriétés sémantiques [55]. Les mêmes types d'arguments sont tenus dans le domaine de la représentation des connaissances, comme en témoigne les deux types de langages de logique de description (basés concepts et basés relations) qui sont utilisés dans le traitement de la langue naturelle [38], une logique de description basée relations pouvant être immergée à une logique de description basée concepts.

Le langage logique de description $\mathcal{ALN}_{r+}$ que nous proposons, est un exemple de ce type d'immersion. Nous nous fonderons sur lui pour la formalisation des compétences ainsi que pour certaines de leurs propriétés sémantiques.

2.2 Découverte de Capacités *(Capability Discovery)*

La découverte de capacités est entendue ici dans un scenario question-réponse. Dit de façon intuitive, une action de découverte de capacités essaie de trouer des capacités qui satisfont une requête. Cette découverte de capacités est fondée sur un modèle de requête sémantique comme dans le modèle général de découverte dans les "affaisres" (*business*) de la figure 4.

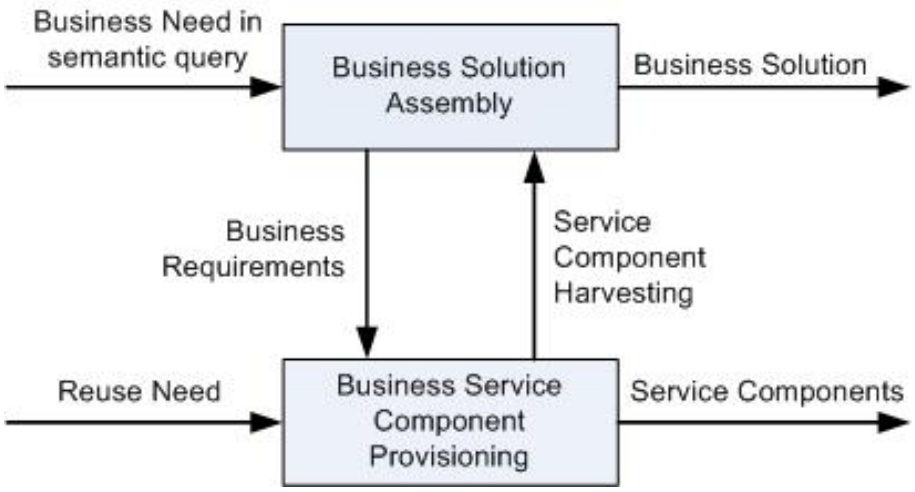


FIG. 4 – Modèle de découverte d'affaires

Un "Business need" est décrit dans langage de reqêtes sémantiques. On obtient un résultat à deux niveaux : solution (*business solution*) et services (*service components*). Le résultat *"business solution"* n'indique pas une ou des compagnies offrant des solutions à la requête mais il décrit le type de compagnies qui satisfairaient la requête. Le résultat *"service components"*, quant à lui, indique quelles compagnies satisfont le *"Business need"*.

De ce fait, la découverte de capacités que nous proposons s'apparente à ce modèle de requête sémantique : elle tente de trouver quels types d'entités possèdent une capacité puis quelles sont ces entités. Par ailleurs, ce processus agit sur une ou plusieurs bases de connaissances décrites dans un langage de représentation de connaissances et, souvent, les bases de connaissances peuvent être distribuées et hétérogènes.

Dans ce travail, nous considérons trois situations de découverte de capacités :

1. *Satisfaction homogène locale* : la requête et la base de connaissances sont exprimées dans le même langage et la base est localisée sur un seul serveur. Dans ce contexte, on considèrera les algorithmes de base pour la satisfaction de requête et l'identification de capacités compémentaires.
2. *Satisfaction homogène distribuée* : la requête et la base de connaissances sont exprimées dans le même langage mais la base de connaissances est constituée de plusieurs bases localisées sur différents serveurs. Dans ce cas, nous considérons la solution du problème dans un cadre distribué.
3. *Satisfaction hétérogène* : les bases de connaissances sont exprimées dans des langages de représentation différents et elles peuvenet également être distribuées. Ctte

situation nous amènera à considérer le problème de l'inter-opération de bases hétérogènes de connaissances.

Plus précisément, parmi les objectifs centraux et les contributions originales de notre travail, nous voulons proposer :

1. un cadre formel pour décrire et organiser des capacités : nous avons opté pour un langage de la famille des langages de logique de description.
2. un concept unique et des mécanismes associés qui satisfont les trois niveaux qui viennent d'être évoqués :
   (a) comparer des entités définies de façon intentionnelle (i.e. des entités définies par un ensemble de capacités),
   (b) identifier des manques en termes de capacités entre entités définies de façon intentionnelle,
   (c) identifier des entités susceptibles de combler les manques,
   (d) contraindre l'ensemble des candidats qui contribue à la satisfaction des manques.

Le concept fondamental sous-jacent à notre travail est celui de *complement*, i.e. l'identification de *"ce qui manque à une entité pour qu'elle soit considérée comme une entité satisfaisant un ensemble donné de capacités"*. Ce concept, très central dans ce travail, apporte les bases appropriées pour, à la fois, *(a)* comparer entités définies de façon intentionnelle, *(b)* identifier les manques éventuels et *(b)* combler ces manques. L'algorithme que nous avons proposé et qui est détaillé dans le chapitre 5 couvre tous ces cas.

Nos contributions sont validées de façon expérimentale par le biais d'une architecture de médiation que nous avons conçue et (partiellement) implémentée.

2.3 Architecture de Médiation

Les médiateurs sont définis comme des modules qui occupent une couche explicite et active entre les applications et leurs ressources [45]. Une fédération de médiateurs peut être requise lorsque une application complexe veut accéder à des données par la biais de divers médiateurs. La fédération fera coopérer les serveurs de médiation indépendants, traduire les lanagges de représentation des médiateurs et intégrer les résultats en une réponse unique homogène.

Dans notre travail, nous explorons les architectures à base de médiateurs pour satisfaire les exigences de la *satisfaction distribuée homogène* aussi bien que la *satisfaction hétérogène* de la découverte de capacités. En effet, dans le premier cas, seule une communication est requise entre médiateurs sitôt que le service qui leur est demandé est identifié : ce service consiste en l'identification d'*entités complémentaires*. Dans le second cas (*satisfaction hétérogène*), une autre fonctionalité est requise : il s'agit de celle qui permet à deux systèmes hétérogènes de se comprendre de façon non ambigue. A cet effet, il est clair que le choix d'un vocabulaire commun et la disponibilité de mécanismes de traduction sont d'une grande aide dans la communications entre différents systèmes de gestion de capacités. Nous abordons cet aspect de façon très pragmatique grâce à l'incorporation de traducteurs de termes dans l'architecture de médiation.

3 Structure et Contenu de la Thèse

La thèse est organisée en sept chapitres et une annexe. Ce qui a précédé constituant l'essentiel du premier chapitre, les six chapitres suivants sont brièvement passés en revue ici.

3.1 Chapitre 2 : Gestion des connaissances et architectures hétérogènes distribuées

Le chapitre 2, en guise de chapitre d'état de l'art, introduit les domaines de recherche auxquels touche cette thèse : la représentation des connaissances, la découverte de capacités et les architecture de médiation. Nous considérons les caractéristiques et les aptitudes de langages de représentaion dans la description de capacités. Nous évoquons également les problème de ré-écriture de représentations de capacités en différents langages.

Le découverte de capacité étant considérée comme une approche de mise en correspondance sémantique (*semantic matching*), nous présentons différents domaines connexes : Intégration sémantique, Mise en correspondance de schémas de données, *Web semantic*, etc.

Enfin, l'architecture de médiation étant une architecture système hybride, hétérogène et distribuée, elle fait référence à des architectures pair-à-pair, les services Web, etc. Ces références sont également introduites dans le chapitre 2.

La figure 5 schématise les points abordés dans le chapitre 2.

3.2 Chapter 3 : Survol de la gestion de capacités dans un environnement hétérogène

Ce chapitre donne une vision conceptuelle de la fédération de médiateurs, de la notion de réponse composée et de la gestion de base de connaissances.

Architecture conceptuelle de médiation et découverte de capacités

D'un point de vue architecture système, nous avons opté pour une architecture à base de négociateurs (ou *trader*, appelé aussi médiateur) [76] très ressemblante à la notion d'agence de découverte (*discovery agency*) dans les architectures pour les services Web. Dans cette architecture, une "entité", appelée exportateur (*exporter*) publie ses capacités auprès d'un ou de plusieurs médiateurs (voir la figure 6). Des entités appelées importateurs (*importers*) envoient des requêtes à un médiateur pour rechercher des exportateurs dotés d'un ensemble donné de capacités.

Les descriptions des capacités sont organisées et rangées chez les médiateurs dans des bases. Ces bases comportent également la descriptions de concepts qui caractérisent les capacités ainsi que les individus qui sont dotés de certaines capacités.

Pour représenter de façon plus formelle les opérations sur les capacités, nous utiliserons des opérations usuelles dans la gestion de bases de donnaissances : *Tell* et *Ask*.

Dans la figure 6 et dans ce qui suit, $\mathcal{L}$ dénote un langage de représentation de connaissances. $\mathcal{L}_{Tell}(E)$ dénote le fait que la description d'une entité E est exprimée dans un

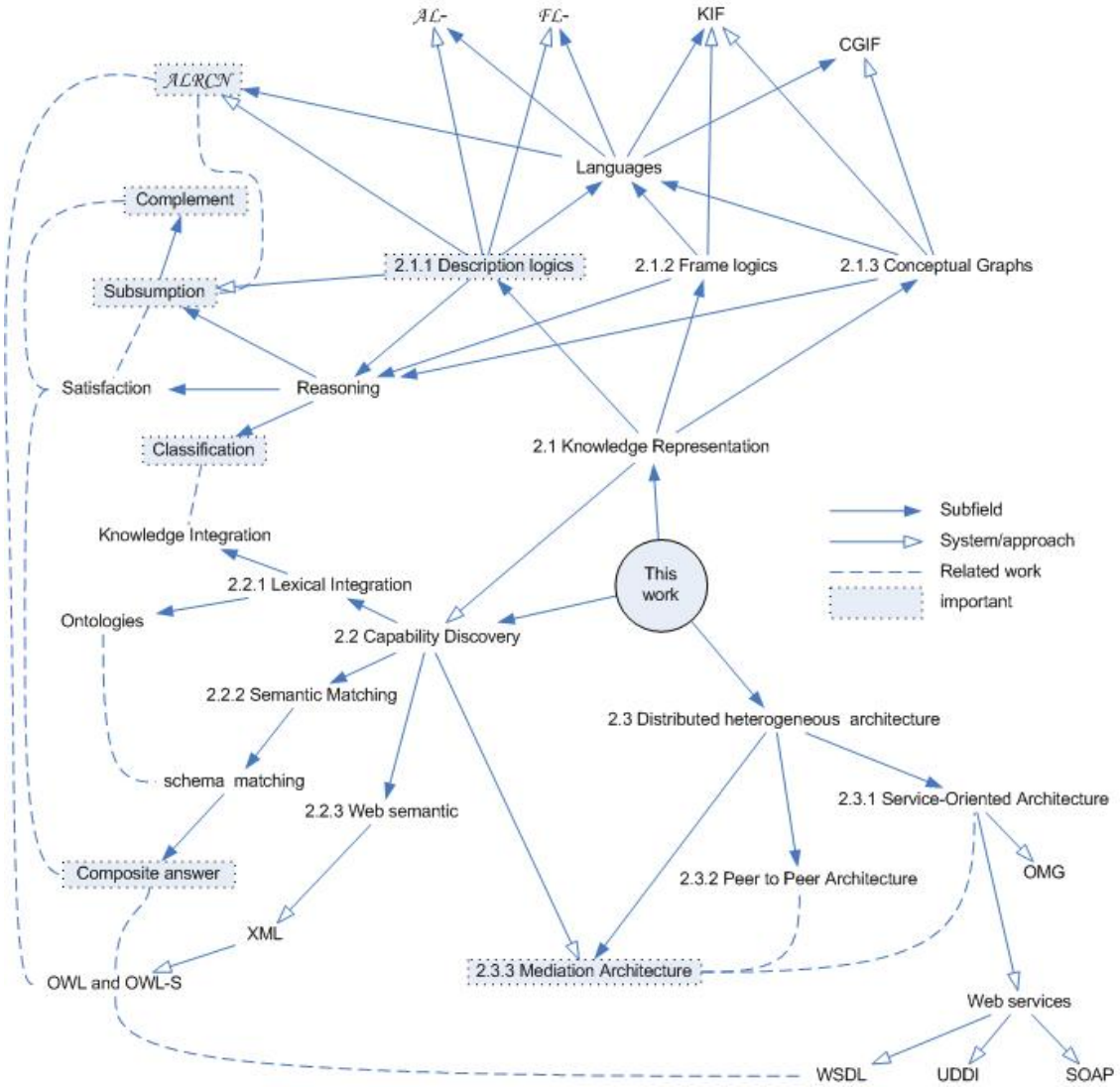


FIG. 5 – "Cartographie" des points abordés dans le chapitre état de l'art

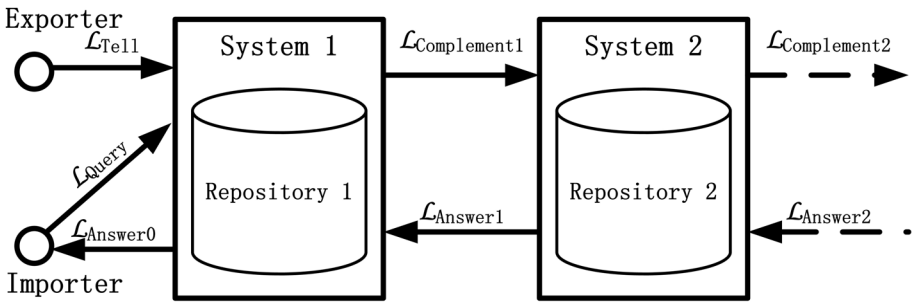


FIG. 6 – Une architecture à base de médiateurs

langage *Tell* donné, $\mathcal{L}_{Tell}$. De façon similaire, $\mathcal{L}_{Query}(Q)$ dénote que la description d'une requête Q est écrite dans un langage de requêtes $\mathcal{L}_{Query}$ et les résultats d'une requête, exprimés dans un langage de réponse $\mathcal{L}_{Answer}$, sont dénotés par $\mathcal{L}_{Answer}(Q)$. Ainsi, *Tell* et une requête (*Query*) peuvent se définir comme suit.

Definition 0.3.1. ***Tell*** *est utilisé pour construire ou pour modifier une base de connaissances KB :*

$$TELL : \mathcal{L}_{Tell}(E) \times KB \rightarrow KB'$$

Definition 0.3.2. ***Ask*** *permet de retrouver des informations, étant donnée une requête Q :*

$$ASK : \mathcal{L}_{Query}(Q) \times KB \rightarrow \mathcal{L}_{Answer}(Q)$$

Dans cette vision conceptuelle, un exportateur utilisera l'opération *Tell* et un importateur utilisera l'action *Ask* avec les deux langages associés : $\mathcal{L}_{Query}$ et $\mathcal{L}_{Answer}$.

Dans cette thèse, nous considérons trois cas de figures pour $\mathcal{L}_{Query}(Q)$ et $\mathcal{L}_{Answer}(Q)$ (voir la figure 7) :

1. la **découverte d'individus** ressemble à une recherche dans une base de données relationnelle. $\mathcal{L}_{Query}(Q)$ ressemble à une formule SQL et le médiateur rend en résultat un ensemble d'objets, cet ensemble étant la description des individus en $\mathcal{L}_{Answer}(Q)$.

2. la **découverte de modèles** "navigue" dans des structures de concepts, où un concept est la modélisation d'un objet. Dans ce travail, les concepts sont structurés en une hiérarchie de subsumption en logique de description. $\mathcal{L}_{Query}(Q)$ est vue comme la description d'un concept et le résultat inclura l'ensemble des concepts qui "correspondent" à la description exprimée par $\mathcal{L}_{Query}(Q)$. En outre, on pourra retrouver des individus représentés par ces concepts : c'est ce que nous appelons l'individualisation de la découverte de concepts. Ainsi, $\mathcal{L}_{Answer}(Q)$ pourra contenir un ensemble de concepts et plusieurs ensembles d'individus.

3. la **découvertes de capacités**, quant à elle, "navigue" dans la structure des capacités, une capacité étant décrite par des relations (appelées *rôles*) entre concepts. $\mathcal{L}_{Query}(Q)$ est alors constituée par la description de caractéristiques de capacités et son résultat inclura l'ensemble des capacités qui possèdent ces caractéristiques : c'est ce qui est appelé la conceptualisation de la découverte de capacités. Toujours selon ce modèle, on peut également "individualiser" les résultats et ainsi obtenir dans $\mathcal{L}_{Answer}(Q)$ un ensemble de capacités, plusieurs ensembles de concepts liés à ces capacités et plusieurs ensembles d'individus liés à ces concepts.

Les propositions pour prendre en compte ces trois situations sont décrites plus en détail dans le chapitre 5 de la thèse et dans ce qui suit, nous proposons une caractérisation de la fédération de médiateurs.

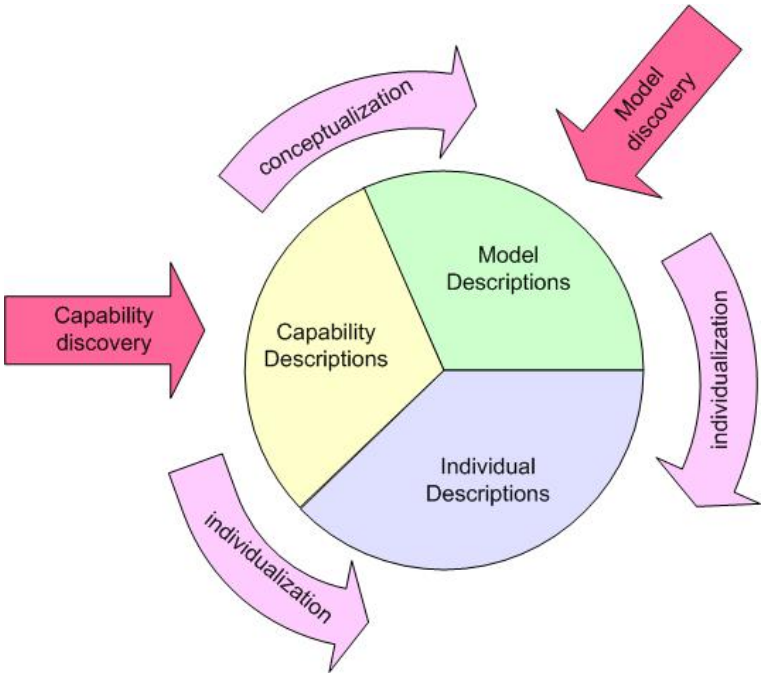


FIG. 7 – Découverte de capacités dans un système $\mathcal{ALN}_{r+}$

Médiateur et fédération Comme déjà dit, un médiateur explore sa base de capacités pour tenter de satisfaire une requête d'un importateur. Outre ce rôle, un médiateur peut avoir d'autres prérogatves comme *(i)* créer ou s'affilier à une fédération, *(ii)* gérer des contextes et *(iii)* propager des requêtes ou des sous-requêtes.

(i) La fédération de médiateurs : elle désigne un groupe de médiateurs qui "acceptent" d'être considérés comme une entité unique par d'autres médiateurs quand ceux-ci demandent un service à la fédération. En d'autres termes, un importateur peut envoyer une demande à un médiateur de la fédération et celle-ci peut être transmise à n'importe quel ou à chaque médiateur de la fédération.

Un médiateur implémente les protocoles $\mathcal{L}_{Query}$ et $\mathcal{L}_{Answer}$. Dans le cadre de la fédération il doit aussi implémenter un protocole nécessaire au support de la notion de complément. Un *complément* (de réponse à une requête) décrit la partie non satisfaite d'une requête et il est exprimé dans un langage $\mathcal{L}_{Complement}$, pour une paire $\mathcal{L}_{Query}$ et $\mathcal{L}_{Answer}$. Comme schématisé sur la figure 6, le *complément* joue le rôle d'une description de requêtes entre deux médiateurs.

(ii) La gestion de contextes dans une fédération :

Les contextes apportent une classification sémantique des connaissances et des données. Au niveau système, chaque médiateur gère et maintient de façon *autonome* ses propres informations (par opposition à une structure centralisée où un médiateur aurait la connaissance des contenus de tous les médiateurs de la fédération).

(iii) La propagation de requêtes dans une fédération :

Une fédération est vue comme un système distribué dynamique dans lequel un médiateur peut coopérer avec des médiateurs différents pour des requêtes différentes. Les

relations entre médiateurs sont alors basées sur la confiance et la crédibilité (*trust and confidence*).

Ainsi, un médiateur peut propager une requête (ou une sous-requête) à un autre médiateur en se fondant sur le "sens" de la requête et l'identification du ou des médiateurs vers lesquels se fait la propagation, peut être opérée :

1. Selon un *critère sémantique* : en "confrontant" la reqête au contexte qu'il gère, le médiateur identifie les médiateurs les plus appropriés pour satisfaire la requête.
2. Selon un *critère de "proximité"* : la requête est propagés vers des médiateurs dèjà identifiés.

Un algorithme de propagation peut évidemment combiner les deux types de critères. Dans notre expérimentation, nous avons choisi une propagation selon le critère de proximité, cet aspect de la fédération n'étant pas central dans nos travaux.

Un modèle pour la compostion de réponses

Notre approche pour la notion de réponses composites est basée sur la notion de satisfaction d'une requête et sur celle de réponse composée.

1. Notion de satisfaction de requête et de réponse composite

La satisfaction d'une requête de recherche de capacités peut s'apparenter aux cas suivants et illustrés sur la figure 8 [20].

- Cas 1 : *Satisfaction exacte* est le cas où il existe des exportateurs qui satisfont exactement la requête, i.e. ils sont dotés des seules capacités requises par la requête.

- Cas 2 : *Satisfaction plus large* est la situation dans laquelle des exportateurs sont dotés de capacités plus grandes que celles recherhées.

- Cas 3 : *Satisfaction par complément(s)* est le cas où aucun exportateur ne répond aux exigences requises mais en "combinant" les capacités de plusieurs exportateurs, on peut satisfaire ces exigences.

- Cas 4 : *Satisfaction partielle* signifie qu'aucun exportateur ni aucune combinaison d'exportateurs ne satisfont la requête.

- Cas 5 : *Echec* est le cas où la requête ne peut pas être satisfaite ni totalement ni partiellement.

Il est important de noter que les cas 4 et 5 (*Satisfaction partielle* et *Echec*, respectivement) conduisent généralement à un échec lors de l'évaluation d'une requête Q quand un seul médiateur est impliqué. Mais dans le cas d'une fédération, ces cas sont des cas typiques de coopération entre médiateurs. Dans le cas 5, toute la requête est transmise à un médiateur de la fédération alors que dans le cas 4, il faudra identifier ce qui "manque" pour la satisfaction de Q, ce "manque" devant ensuite être envoyé comme une sous-requête

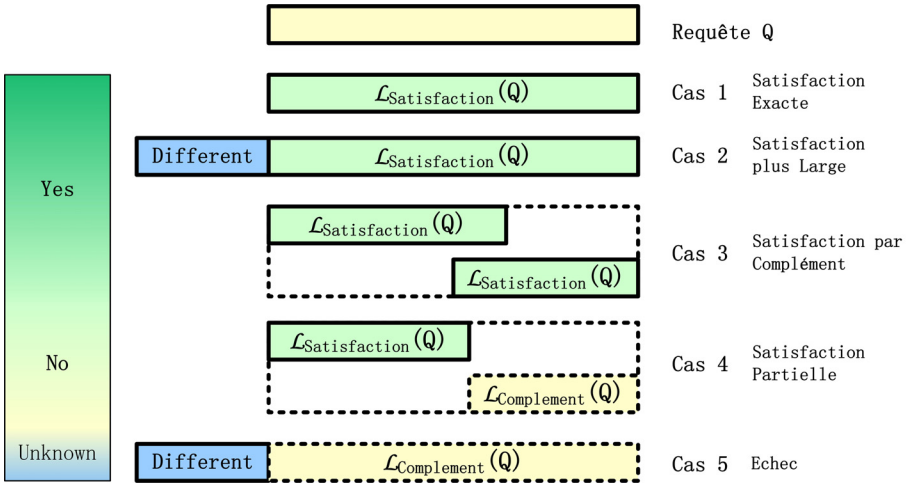


FIG. 8 – Satisfaction de requêtes : les différents cas

à un autre médiateur. Conceptuellement, on peut voir la requête comme étant adressée à "l'union" des bases de capacités gérées par les mediateurs fédérés, cette "union" étant explorée de proche en proche, c'est-à-dire un médiateur après l'autre selon un ordre défini (cas de la propagation par proximité) ou un ordre calculé (cas de la propagation selon un critère sémantique).

De ce fait, une réponse à une recherche de capacités peut être composée de plusieurs "fragments" de réponses. Par ailleurs, il peut exister plusieurs réponses candidates. Aussi une stratégie de choix de ces réponses candidates peut parfois être imposée dans le processus d'évaluation de la requête ou des contraintes sur le nombre de "fragments" de réponses peuvent être exprimées par l'importateur.

2. Notion de contrainte sur les réponses composites

En fait, la figure 8 montre une stratégie *"premier venu"*, qui est celle que nous avons appliqué dans notre prototype. Cette stratégie essaie toujours de trouver une satisfaction totale (exacte, plus large ou par complément). Mais on peut parfois vouloir contraindre le type de réponse souhaité. Par exemple, on peut vouloir imposer que la réponse ne doit pas être composée de plus de deux "fragments" et en cas de satisfaction plus large, les capacités supplémentaires non requises doivent être les plus petites possibles[1].

La façon dont cette notion de stratégie est prise en compte dans notre proposition est détaillée dans le chapitre 5 (section 5.2, page 100) où nous présentons l'algorithme de calcul effectif du complément.

Chapitre 3 : conclusion

Ce chapitre résume la vision conceptuelle de notre problématique et de sa solution. Les chapitres suivants détaillent successivement nos propositions,

[1]D'un point de vue application, ce type de contraintes permet d'exprimer des requêtes du type, par exemple, recherche d'au plus x partenaires.

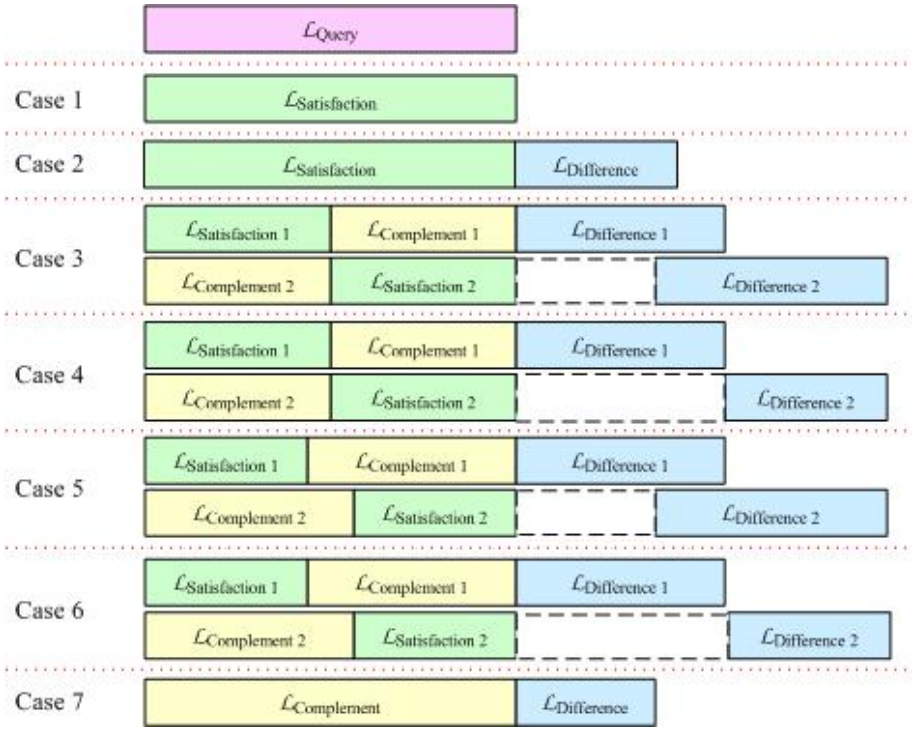


FIG. 9 – Cas d'une réponse composée limitée à deux fragments de réponse

- en termes de langage formelle de représentation de capacités (chapitre 4),
- en termes de processus et d'algorithmes de découverte de capacités tant dans des bases homogènes (i.e. toutes exprimées dans le langage de représentation de capacités proposé) qu'hétérogènes, i.e. en présence de bases exprimées dans des langages de représentation de capacités différents (chapitre 5),
- en termes de choix faits lors de l'implémentation d'un prototype de médiateurs fédérés (chapitre 6).

3.3 Chapitre 4 : $\mathcal{ALN}_{r+}$, le langage de représentation de capacités

Dans ce chapitre nous appliquons une partie de la théorie de la représentation des connaissances à la gestion de capacités. Nous proposons un langage de description $\mathcal{ALN}_{r+}$ qui étend le langage logique de description (DL) $\mathcal{ALN}$. Le modèle de gestion de capacités en $\mathcal{ALN}_{r+}$ dérive des systèmes clasiques de DL qui distinguent une base terminologique (TBox) et une base d'assertions (ABox). Pour notre part, nous ajoutons une base T$_r$Box distincte de la TBox pour y ranger les connaissances relatives aux capacités, connaissances exprimées à l'aide du concept de rôle en DL.

Dans la description de la syntaxe de $\mathcal{ALN}_{r+}$ de la figure 10, les conventions usuelles de notation sont utilisées :

- les lettres A, B sont utilisées pour des *concepts primitifs* et C, D pour les *concepts définis* (à l'aide de concepts primitifs et/ou de concepts définis)
- de façon similaire, r, s sont utilisés pour dénoter des *rôles primitifs* et R, S pour des

rôles définis,
- f, g désignenent des restictions fonctionnelles sur les rôles,
- a, b, c, d dénotent des individus.

Dans cette figure, la partie haute concerne la description de concepts et la partie basse celle des rôles.

Nom	Syntaxe abstraite		Syntaxe Concrète
primitive concept	$C, D \rightarrow$	$A \mid$	`A`
universal concept		$\top \mid$	`TOP`
bottom concept		$\bot \mid$	`BOTTOM`
primitive negation		$\neg A \mid$	`(not A)`
at-least restriction		$(\geq\ n\ r) \mid$	`(atleast n r)`
at-most restriction		$(\leq\ n\ r) \mid$	`(atmost n r)`
concept conjunction		$C \sqcap D \mid$	`(and C D)`
value restriction on roles		$\forall R.C$	`(all R C)`
role name	$R, S \rightarrow$	$r \mid$	`r`
universal role		$\top_{role} \mid$	`TOProle`
bottom role		$\bot_{role} \mid$	`BOTTOMrole`
role disjunction		$R \cup S \mid$	`(or R S)`
symmetric closure		$R^{-} \mid$	`(R`$^{-}$`)`
transitive closure		$R^{+} \mid$	`(R`$^{+}$`)`
reflexive-transitive closure		$R^{*} \mid$	`(R`*`)`
role functional restriction		RfS	`(R`f`S)`

FIG. 10 – Syntaxe de $\mathcal{ALN}_{r+}$

Outre cette définition syntaxique du langage $\mathcal{ALN}_{r+}$, ce chapitre comprend la définition de sa sémantique ainsi que les définitions de concepts importants pour la suite des travaux et leurs fondements, à savoir, les notions de composition, d'équivalence, de subsumption et de restriction fonctionnelle de rôles.

3.4 Chapitre 5 : Gestion et découverte de capacités dans une base $\mathcal{ALN}_{r+}$

Dans ce chapitre, nous présentons la théorie et les algorithmes de raisonnement dans un système de représentation de connaissances en $\mathcal{ALN}_{r+}$. Ces algorithmes sont fondés sur l'identification des relations de subsumption entre deux structures de description de capacités. En suivant une démarche par normalisation-comparaison, les deux stuctures sont d'abord mises sous forme normale avant de pouvoir être comparées.

Ce chapitre détaille également notre approche pour la découverte de capacités. Comme déjà exposé, celle-ci se base sur le modèle de réponses composites qui comprennent une partie satisfaite et une partie à satisfaire appelée complément. La partie satisfaite est

déterminée grâce à la relation de subsumption et la partie complément l'est grâce à une implémentation procédurale du *concept de complément* [91].

Dans le chapitre introductif à ce travail, nous avons distingué la *gestion de capacités* et la *découverte de capacités* : l'approche et les algorithmes pour ces deux aspects sont détaillés dans ce chapitre..

La *gestion de capacités* (développée dans le paragraphe 5.1) se base sur deux hiérarchies : une hiérarchie de concepts et une hérarchie de rôles, ces hiérarchies étant bâties grâce à la relation de subsumption entre concepts et entre rôles, respectivement, et elles sont construites et maintenues grâce à un mécanisme de classification. Ce mécanisme étant lui-même fondé sur la relation de subsumption, nous montrons comment celle-ci est calculée selon le processus normalisation-comparaison (section 5.1.2).

En ce qui concerne la *découverte de capacités*, elle utilise les hiérarchies de la *gestion de capacités* pour évaluer les requêtes qui sont adressées à la base $\mathcal{ALN}_{r+}$. Dans notre proposition, une requête est vue comme un concept à "situer" (classifier) dans les hiérarchies disponibles.

A l'aide des concepts et des mécanismes définis, nous montrons comment nous apportons une solution uniforme aux cas évoqués dans la figure 8. Cette solution est résumée ci-dessous.

Réponse Composite dans$\mathcal{ALN}_{r+}$

D'un point de vue logique, une requête est un calcul de satisfiabilité (cf. définition 3.2.2, page 67). Un concept subsumé satisfait toujours le concept subsumant dans $\mathcal{ALN}_{r+} - \mathcal{T}$, c'est-à-dire que le concept subsumant représente la requête, ses subsumés constituent une réponse qui satisfait totalement la requête. Néanmoins, avant de procéder au calcul de réponses composites, on se doit de déterminer ses deux composants de base : la partie satisfaite et la partie à satisfaire (que nous appelons *complément*).

Nous utilisons une description C de concept pour exprimer une requête : alors, si on peut trouver une description D de concept qui vérifie $D \sqsubseteq C$, on pourra en conclure que D satisfait totalement la requête. Pour ce faire, les descriptions de concepts sont mises sous forme normale conjonctive avant d'être comparées. Ainsi, une description C de concept resprésentant une requête Q sera exprimée sous la forme $C_1 \sqcap \cdots \sqcap C_n$ et une description D de concept candidate à une réponse à Q sera exprimée sous la forme $D_1 \sqcap \cdots \sqcap D_n$. Dans l'exemple de la figure 11, $C_1, \ldots, C_k$ et $D_1, \ldots, D_k$ sont supposés satisfaire la relation de subsumption. Donc, selon notre notation, $\mathcal{L}_{Satisfaction}(Q) = D_1 \sqcap \cdots \sqcap D_k$.

Les structures atomiques $C_{k+1}, \ldots, C_n$ n'étant pas satisfaites, elles constitueront la description du complément qu'il faudra déterminer : $\mathcal{L}_{Complement}(Q) = C_{k+1} \sqcap \cdots \sqcap C_n$.

A titre d'exemple, considérons le concept `CITY-AIRPORT` qui caractérise des villes dotées de capacités de transport aérien, et le concept `PORT-AIR` qui caractérise les villes dotées de capacités de transport maritime et aérien. Les formes normales de ces deux concepts sont données ci-dessous. Supposons que `PORT-AIR` soit la requête (dénotée C ci-dessous) et `CITY-AIRPORT`, dénoté D, soit un concept de la bas de connaissances.

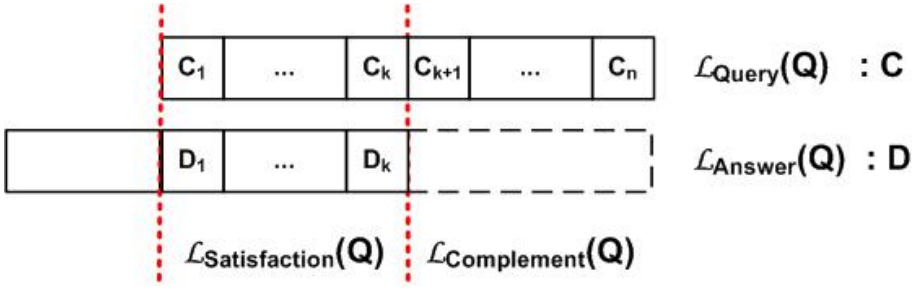


FIG. 11 – Détermination de réponse composite

$$
\begin{array}{lll}
C: \texttt{PORT-AIR} = & (\sqcap\ \texttt{CITY} & (C_1) \\
& \quad (\geq\ \infty\ 1\ \texttt{has-flight}^{-}).\texttt{CITY}) & (C_2) \\
& \quad (\geq\ \infty\ 1\ \texttt{has-ferry}).\texttt{CITY})) & (C_3) \\
 & & \\
D: \texttt{CITY-AIRPORT} = & (\sqcap\ \texttt{CITY} & (D_1) \\
& \quad (\geq\ \infty\ 1\ \texttt{has-flight}^{-}).\texttt{CITY})) & (D_2)
\end{array}
$$

Dans ces descriptions, $D_1 \sqcap D_2$ constitue la partie satisfaction partielle de C et C_3 constitue la partie complément à satisfaire.

Détermination d'un complément pour une requête

La détermination du complément de réponse pour une requête s'effectue en deux étapes :

1. Identifier le concept complément , i.e. la partie de la requête qui n'est pas satisfaite ;
2. Trouver effectivement le complément, s'il existe, dans la base de connaissances.

Ces deux étapes sont successivement décrites ci-après.

A. Identification du complément : Pour identifier la partie satisfaite et celle non satisfaite en réponse à une requête, nous introduisons une table de booléens que nous appellons table de satisfaction (abrégé en ST, ci-après). ST contiendra les résultats de l'évaluation de la relation de subsumption. Ainsi, dans la figure 12, $C_1, C_2, \ldots, C_n$ dénotent la requête sous forme normale et $D_1, D_2, \ldots, D_m$ dénotent des concepts sour leur forme normale également où chaque D_j doit être vue comme une conjonction $(and D_j^1, D_j^2, \ldots, D_j^{nj})$ de concepts atomiques. Ainsi,

$$ST[D_j, C_i] = true \leftrightarrow\ D_j^i \sqsubseteq C_i$$

A des fins d'implémentation, une fonction *Subsume(Subsumant, Subsumé)* permet de tester si la relation de subsumption est vraie entre ses arguments (*Subsumant* et *Subsumé*). Quand la valeur rendue par $Subsume(C, D_j)$ est "False" (c'est-à-dire que le concept D_j

ne satisfait pas le concept C), il faudra alors déterminer un complément éventuel de D_j relativement à C. Cette détermination est fondée sur la table de satisfaction ST.

Dans l'exemple précédent concernant `PORT-AIR` et `CITY-AIRPORT`,

$$Subsumes(\texttt{PORT-AIR}, \texttt{CITY-AIRPORT}) = \texttt{false},$$

i.e. `CITY-AIRPORT` $\neg \sqsubseteq$ `PORT-AIR`, et `CITY-AIRPORT` ne staisafit pas totalement `PORT-AIR`, "($\geq \infty$ `1 has-ferry`).CITY" étant la partie *Comp* non satisfaite :

$$Comp(\texttt{PORT-AIR},\texttt{CITY-AIRPORT}) = (\geq \infty \texttt{ 1 has-ferry}).\text{CITY}.$$

De façon plus générale, le complément dénoté $Comp(Requete, Satisfaction)$ est défini par :

Definition 0.3.3. *$Comp(C, D) = \sqcap_{k=1}^{r} C_k$ for every k, $k \in [1..n]$ such that $ST[k] = false$.*

En d'autre termes, le complément est donné par la conjonction des concepts atomiques pour lesquels la valeur correspondante dnas la table de satisfaction ST est égale à "False".

	C_1	C_2	...	C_n
D_1	False	False	...	True
D_2	False	True	...	True
...	...	...	...	...
D_m	False	False	...	False
ORoD	False	True	...	True

ORoS	ANDoS
True	False

FIG. 12 – Les trois paramètres de décision de la table de satisfaction : ORoD, ORoS and ANDoS

B. Calcul du complément : Montrons d'abord le principe du calcul effectif du complément avant de généraliser le discours en termes d'algorithme.

B.1. Calcul du complément, principe et exemples : Pour essayer de calculer effectivement le complément identifié, nous introduisons trois variables booléennes comme paramètres de décision : ORoD, ORoS et ANDoS (figure 12). Ces paramères servent à déterminer les conditions de satisfaction des cas introduits dans le paragraphe 3.2 (page 12) et rappelés ci-dessous :

1. $ORoD[1..n]$ est défini par $ORoD[i] = \bigvee_{j=1}^{m} ST[D_j^i, C_i], \forall i \in [1..n]$, i.e. $ORoD[i]$ est la disjonction des valeurs de vérité de la colonne i de la table ST et s'interprète de la façon suivante : $ORoD[i] = true$ signifie que le concept C_i de la requête C est satisfait par au moins un des concepts $D_k^i, k \in [1..m]$.

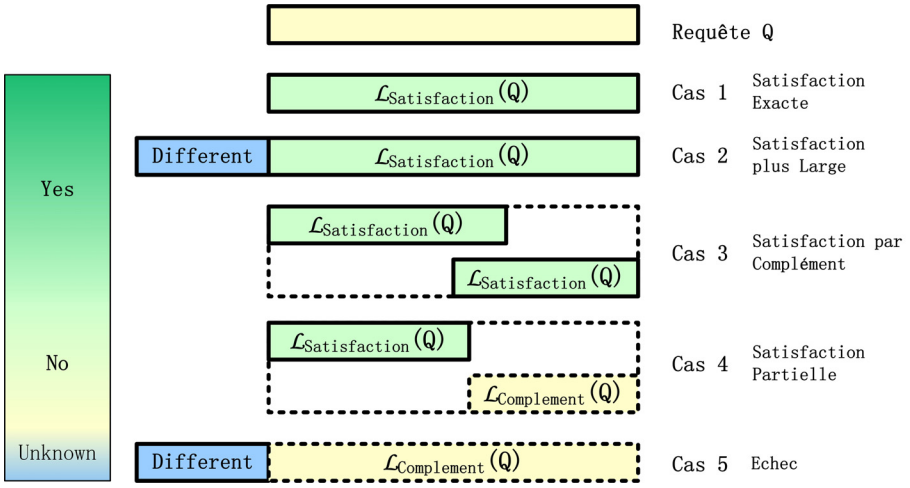


FIG. 13 – Satisfaction de requêtes : les différents cas

2. De façon similaire, si la conjonction des valeurs de $ORoD$, dénotée par $ANDoS$ et définie par $ANDoS = \bigwedge_{i=1}^{n} ORoD[i]$, s'évalue à vrai, ceal signifie que tous les concepts C_is de la requête C sont satisfaist et, par conséquent, C est satisfaite.
3. Finalement, quand $ANDoS$ est évaluée à faux, la disjonction logique des valeurs de $ORoD$, dénotée $ORoS$ et définie par $ORoS = \bigvee_{i=1}^{n} ORoD[i]$, permet de déterminer les cas de satisfaction partielle et celui d'échec.
   - Si $ANDoS$ est faux et que $ORoS = True$ est vrai, cela signifie qu'il existe un C_k de la requête C qui n'est pas satisfait ;
   - Si $ANDoS$ et $ORoS = True$ sont tous deux évalués à faux alors aucun concept atomique D_j^k parmi les concepts D_j ($j \in 1..m$) ne satisfait un des concepts C_i de C.

Ainsi, à l'aide des résultats de la classification, qui identifie le plus petit commun subsumant (*Least Common Subsumer (LCS)*) et le concept le plus spécifique (*Most Specific Concept (MSC)*), et de ceux de l'algorithme de Normalisation-Comparaison, qui rend les valeurs des paramètres de décision $ORoS$ et $ANDoS$, les différents cas de satisfaction peuvent être couverts : la table 1 synthétise cette discussion. Dans cette table, X et Y sont des descriptions de concepts, $\top$ est le concept le plus général et $\bot$ le concept le plus spécifique.

LCS(Q)	MSC(Q)	ORoS	ANDoS	CAS
X	Y	True	True	1 : Satisfaction exacte
X	$\bot$	True	True	2 : Satisfaction plus large
$\top$	$\bot$	True	True	3 : Satisfaction par complément
$\top$	$\bot$	True	False	4 : Satisfaction partielle
$\top$	$\bot$	False	False	5 : Echec

TAB. 1 – Analyse des cas et des conditions de satisfaction

Les conditions de satisfaction sont exprimées formellement comme suit :

1. *Satisfaction exacte* : $Y \sqsubseteq Q \sqsubseteq X$ iff $(LCS(Q) = X) \wedge (MSC(Q) = Y) \wedge (ORoS = True) \wedge (ANDoS = True)$;
2. *Satisfaction plus large* : $Q \sqsubseteq X$ iff $(LCS(Q) = X) \wedge (MSC(Q) = \bot) \wedge (ORoS = True) \wedge (ANDoS = True)$;
3. *Satisfaction par complément* : $\exists X_1, \ldots, \exists X_n, Q \sqsubseteq \bigcup_{i=1}^{n} X_i$ iff $(LCS(Q) = \top) \wedge (MSC(Q) = \bot) \wedge (ORoS = True) \wedge (ANDoS = True)$;
4. *Satisfaction partielle* : $\exists X_1, \ldots, \exists X_n, \bigcup_{i=1}^{n} X_i \sqsubset Q \wedge \neg(\exists Y_1, \ldots, \exists Y_m, Q \sqsubseteq \bigcup_{j=1}^{m} Y_j)$ iff $(LCS(Q) = \top) \wedge (MSC(Q) = \bot) \wedge (ORoS = True) \wedge (ANDoS = False)$;
5. *Echec* : $\forall X \in \mathcal{T}, X \cap Q = \emptyset$ iff $(LCS(Q) = \top) \wedge (MSC(Q) = \bot) \wedge (ORoS = False) \wedge (ANDoS = False)$.

B.2. Algorithme de calcul du complément de requête : D'un point de vue implémentation, une description de concept sous forme normale est représentée dans la table de satisfaction comme un tableau $T[1..n]$ de variables booléennes où :

1. la taille n du tableau est le nombre de descriptions de concepts atomiques dans la description de la requête en tant que concept,
2. les valeurs des entrées de ce tableau indiquent quels concepts atomiques sont satisfaits ou pas, selon le résultat rendu par l'algorithme de test de la relation de subsumption.

Dans la cas où plusieurs descriptions de concepts sont candidates à la satisfaction de la description de la requête, elles peuvent être ordonnées selon un critère donné (appelé aussi *priorité*) pour choisir celles à considérer en premier pour valuer la table de satisfaction. Ce critère peut être simple (comme considérer en premier les descriptions de concepts qui satisfont le plus grand nombre de concepts de la requête) ou plus complexe, comme *(i)* "les concepts homogènes d'abord puis le plus grand nombre de concepts satisfaits" ou encore *(ii)* "les concepts C_i and C_j doivent être satisfaits par une entité unique", etc.

L'algorithme peut ainsi donner lieu à diverses implémentations. Celui schématisé sur la figure 14 utilise le critère de priorité qui consiste à ordonner les concepts candidats seleon le nombre (décroissant) de concepts satisfaits par chacun d'eux. Dans cet algorithme,

- `Entities[1..n]` est la représentation sous forme de table de booléens d'une description de concept candidate,
- `ST` est la table de satisfaction sur laquelle sont définies des méthodes dont :
  - `ST.clear` qui crée et initialise la table de satisfation ;
  - `ST.add()` qui ajoute une ligne à cette table ;
  - `ST.DeleteLastLine` qui supprime une ligne de la table.

Les valeurs des paramètres de décision `ORoD`, `ORoS` et `ANDoS` sont modifiées par l'invocation de chacune de ces opérations et si la valeur de `ORoD` venait à être modifiée, la variable indicateur `ORoD_change` l'indiquera (mise à vrai). La figure 14 montre aussi le point où une stratègie différente peut être intégrée : par exemple, le fait d'ajouter une limitation du nombre de lignes de la table de satisfaction (en ajoutant `(and ST.index < 2)` dans le test de la variation de la valeur de `ORoD_change`), permettra de limiter à 2 le nombre de fragments composant une réponse.

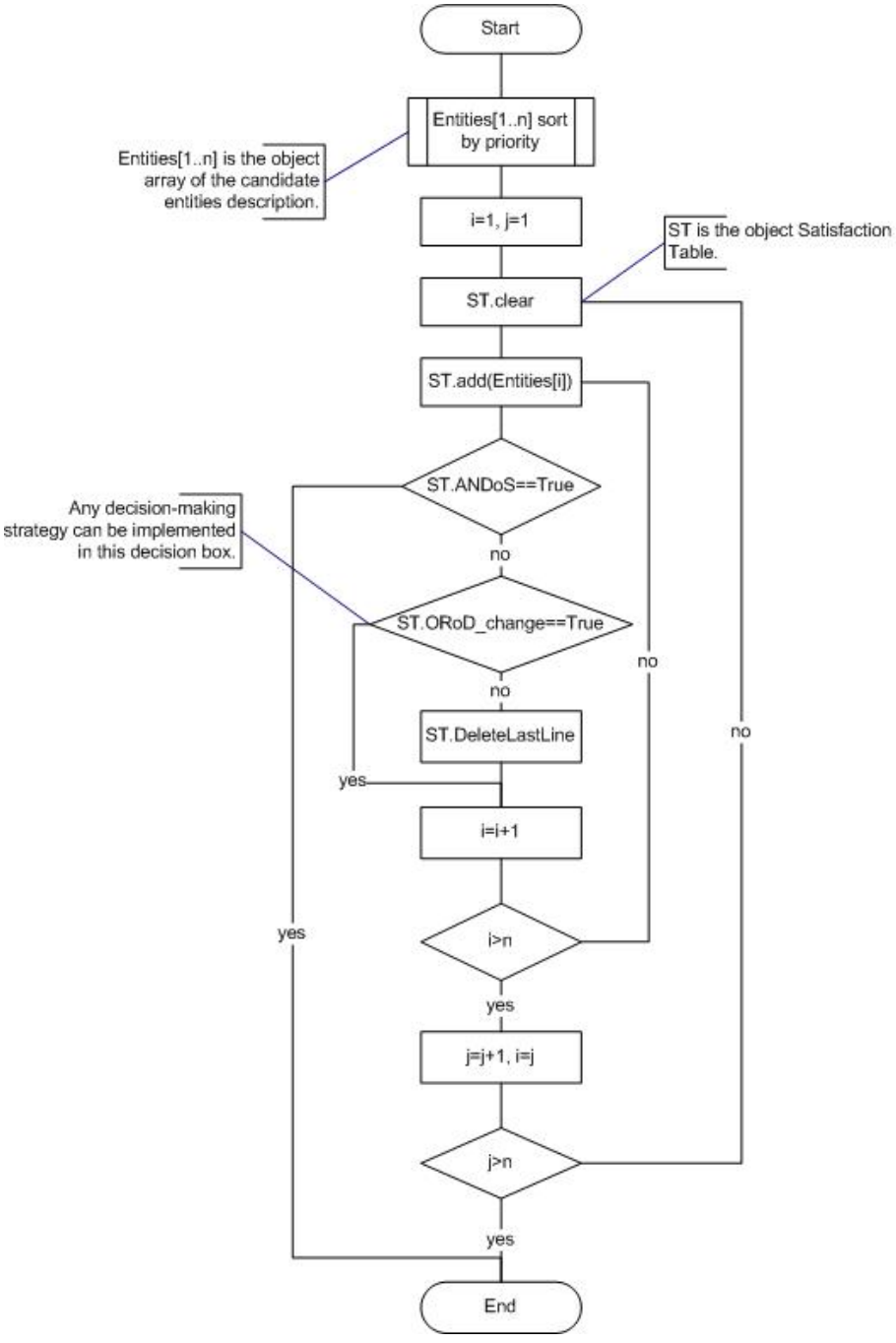


FIG. 14 – Organigramme de l'algorithme de gestion de la *Table de Satisfaction*

B.3. Conclusion : Le discours qui précède s'applique au cas où une requête est évaluée par un seul médiateur. Comme déjà évoqué, les cas de satisfaction partielle et d'échec peuvent amener à faire appel à d'autres médiateurs pour tenter de pallier l'insatisfaction totale ou l'échec. C'est ce que nous présentons dans le paragraphe qui suit.

Découverte de capacités par de multiples médiateurs

Dans notre introduction à ce travail, nous avions distingué trois situations :

1. *Satisfaction homogène locale* où un seul médiateur peut satisfaire une requête,
2. *Satisfaction homogène distribuée* où plusieurs médiateurs utilisant le même langage de représentation de connaissances se "complémentent" pour apporter une solution,
3. *Satisfaction hétérogène* où plusieurs médiateurs utilisant des langages de représentation différents coopèrent pour essayer de satisfaire une requête.

Le premier cas vien d'être traité. Ce qui distingue les deux derniers cas est le fait qu'un mécanisme de traduction est requis quand (cas 3) les langages de description de capacités sont différents. Néanmoins, un mécanisme additionnel est requis dans les deux cas lorsque les médiateurs n'utilisent pas un vocabulaire commun : dans cette situation, il est nécessaire d'opérer une traduction de descriptions des concepts ainsi qu'une correspondance (*matching*) entre les termes des vocabulaires utilisés.

La traduction des descriptions est opérée sur des descriptions atomiques de concepts. Pour notre expérimentation, nous avons étudié les correspondances entre des description de concepts exprimées en logique de description, en logique de *frames* [44] et en graphes conceptuels [64]. Quant à la correspondance entre termes du vocabulaire, nous opérons une traduction lexicale fondée sur une évaluation de la similitude entre les termes.

Dans le chapitre 5 de la thèse, ces considérations sont détaillées de la façon suivante (paragraphe 5.2.3, page 109) :

1. Nous expliquons la notion de description de concept atomique et nous présentons les diverses formes de description que nous supportons (page 109),
2. nous introduisons la traduction lexicale de termes que nous avons adoptée (page 110),
3. nous présentons le notion de réponse composite dans un cadre hétérogène (page 111),
4. et nous montrons le processus de calcul du complément dans ce type de cadre (page 113).

Conclusions du chapitre 5

Ce chapitre détaille le cœur de notre travail où une base de connaisances en $\mathcal{ALN}_{r+}$ est constituée de trois parties : TBox, T$_r$Box et ABox.

Nous avons opté pour une approche par classification pour la maintenance les hiérarchies de concepts et de rôles. Cette approche utilise un algorithme de normalisation-comparaison pour le test de la relation de subsumption. Le résultat de cet algorithme étant ensuite utilisé dans le calcul de réponses composites. Cet algorithme est correct et complet dans $\mathcal{ALN}_{r+}$, mais il ne s'applique pas à tous les langages de description logique. D'autres algorithmes, comme celui des tableaux [6], pourraient aussi être utilisés pour ces calculs.

D'un point de vue implémentation, les résultats de la comparaison sont mis dans une table de satisfaction dont l'analyse permet de déterminer une réponse composite.

Nous avons étendu notre proposition au cas hétérogène dans lequel la traduction de formules entre langages hétérogènes est, pour l'heure, opérée manuellement. Par ailleurs, nous avons étayé nos propos en nous fondant sur un exemple de "capacités à transporter" des individus. Mais il est clair que notre approche peut s'appliquer aisément à d'autres domaines et, à titre d'illustration, nous annexons à ce document un exemple basé sur les connaissances dans le domaine de l'interopérabilité des sytèmes et des logiciels.

Enfin, nous avons développé un prototype pour vailder nos propositions : le chapitre 6 détaille l'architecture, les fonctionalités et les choix d'implémentation de ce prototype.

3.5 Chapitre 6 : Fédération de médiateurs

Le prototype développé consiste en deux parties *(i)* l'application de la gestion et de la découverte de capacités dans une base $\mathcal{ALN}_{r+}$ et *(ii)* la composition de serveurs de médiation dans un environnement hétérogène.

Dans la première partie de ce prototype, toutes les fonctionalités requises par ce qui a été exposé dans le chapitre précédent, et celles requises par la gestion et la découverte de capacités, ont été développées (en *Java*). Ces fonctionalités incluent :

- un service pour le test de la relation de subsumption,
- un service pour la détermination du *complément* et la *satisfaction* de concepts,
- et un service pour le calcul de *réponses composites*.

Dans la seconde partie du prototype, nous avons expérimenté les services de médiation composites dans un environnement hétérogène, comme montré sur la figure 15 où :

- Mediator 1 "comprend" le langage $\mathcal{ALN}_{r+}$,
- Mediator 2 "comprend" la langage logique de *frames* (F-Logic) [44] : son langage de communication est KIF [41] ;
- Mediator 3 "comprend" les graphes conceptuels (CGs) : son langage de communication est CGIF [64].

Cette architecture fédérée a été développée en conformité avec les architectures orientées services (Service Oriented Architecture ou SOA) [5].

Par ailleurs, chaque serveur de médiation comporte quatre composants (figure 15) :

1. une base locale,
2. un raisonneur,
3. un traducteur syntaxique,
4. et un dictionnaire ontologique lexical.

3.6 Chapitre 7 : Conclusions, bilan et perspectives

Dans ce travail, nous considérons avoir atteint deux résultats essentiels :

1. la définition d'un langage de description de capacités, $\mathcal{ALN}_{r+}$, pour supporter la gestion de capacités et les services d'inférence inhérents.

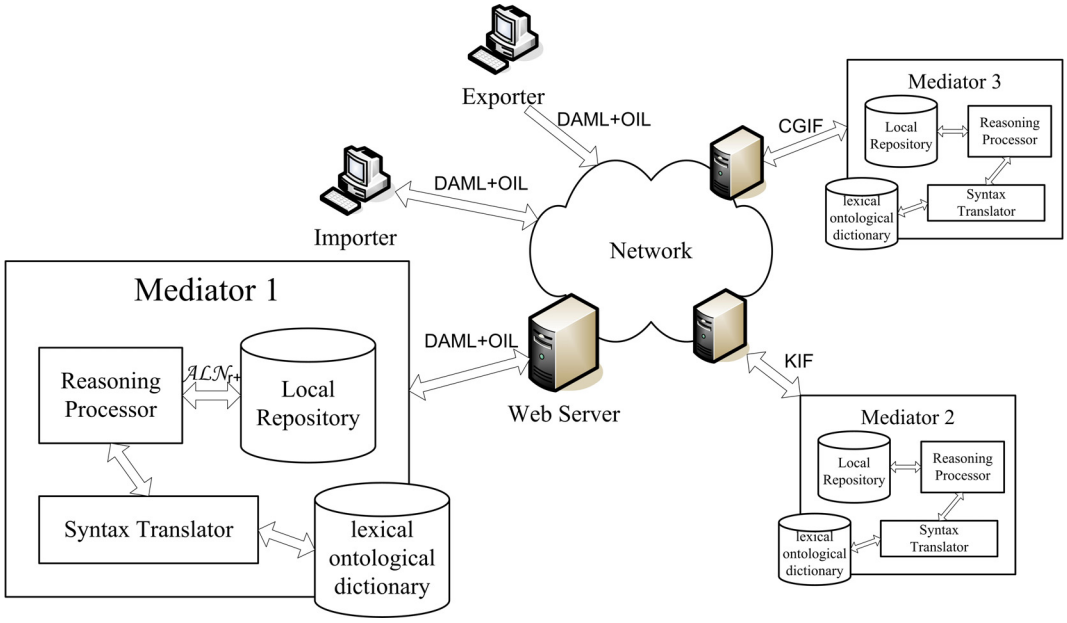


FIG. 15 – Architecture de Médiation

2. la conception (et l'implantation partielle) d'un système de médiation fédéré qui permet la composition de réponses lors de la recherche de capacités dans des environnments hétérogènes de représentation de connaissance.

A propos du premier résultat et considérant les objectifs de cette thèse tels qu'ils sont exposés dans le paragraphe 2.2

1. *Du point de vue description et organisation des capacités* : $\mathcal{ALN}_{r+}$, le langage proposé, s'appuie sur la notion de *rôle* (i.e. relations entre concepts) pour la description formelle de capacités possédées par des "entités". $\mathcal{ALN}_{r+}$ hérite et étend la théorie et la technologie des langages logiques de description [6]. Nous avons implémenté des services d'inférence basés sur ce langage qui servent tant pour la gestion de capacités que pour leur découverte. Nous avons aussi introduit un modèle ouvert de restriction fonctionnelle f entre rôles (i.e. de relations entre des relations de concepts) pour décrire les relations entre des rôles et nous avons introduit un modèle très simple de description de relations entre rôles dans le langage de description de capacités $\mathcal{ALN}_{r+}$. En outre, dans ce travail, nous avons considéré plusieurs relations entre capacités, dont la *subsumption*, la *composition* et l'*equivalence* de rôles.
2. *Du point de vue découverte de capacités* : nous nous sommes appuyés sur un seul concept (le concept de complément) et sur sa détermination procédurale pour satisfaire des facilités requises par la gestion de capacités, de compétences mais aussi de connaissances. Ces facilités permettent plus précisément :
   (a) *de comparer des entités définies de façon intentionnelles*, i.e. deux ensembles d'individus caractérisés par leur formule en $\mathcal{ALN}_{r+}$ peuvent être comparés à l'aide des mécanismes proposés :

  - Les deux ensembles sont considérés comme "égaux", en termes de capacités, quand la formule qui décrit un ensemble satisfait de façon exacte la formule qui décrit le second ensemble.
  - Un ensemble est "plus grand" qu'un autre, toujours en termes de capacités, quand une formule *subsume* l'autre (voir la notion de satisfaction plus grande)

(b) *d'identifier des capacités manquantes* grâce à l'identification du complément,

(c) *de combler le manque en capacités* en calculant le complément

(d) *de contraindre les ensembles de candidats* pour combler les manques, par le biais de stratégies qui peuvent contraindre le processus de calcul du complément.

Concernant le second résultat, nous avons conçu et implémenté un prototype de système de médiation fédéré. Le prototype est totalement écrit en Java (Sun Microsystem's JDK1.2 et Java Web Services Developer Pack 1.6) et il a été testé sous divers systèmes d'exploitation : Miscrosoft Windows (95, 98, Me, XP, 2000, 2003) and Linux (Red Hat 7.X and Mandrake 8.x). En outre, le prototype a une architecture basée sur les standards pour les services Web et il a une architecture hybride. Un importateur, un exportateur et un serveur de médiation composent un système typique orienté services (*Service Orented Architecture*). Notre prototype se positionne comme suit par rapport aux caractéristiques d'un système pair-à-pair (introduites dans le chapitre 2, section 2.3.2, page 55) :

1. *Du point de vue autonomie* : l'autonomie est atteinte du fait que chaque serveur de médiation offre les services (*Ask* and *Tell*) à ses clients (importateur et exportateur, respectivement) ;
2. *Du point de vue dynamicité* : le prototype dépend d'un document OWL-S pour la gestion de toute la fédération de médiateurs. Cette solution n'est clairement pas adaptée pour des applications "réelles". UDDI [100] définit comment des entités publient leurs services et se découvrent les unes les autres. UDDI permet également de définir comment des services ou des applications logicielles inte-agissent sur internet. Ce principe peut être appliqué dans l'architecture de fédération pour implémenter la gestion et la découverte dynamique de médiateurs et de leurs services. Cependant, cette propriéte de la fédération requiert d'autre mécanismes pour la gestion concrète de la fédération, i.e. des mécanismes permettant à un médiateur de rejoindre ou de quitter (temporairement ou définitivement) une fédération. Cet aspect n'a pas été considéré dans ce travail car il n'était pas central dans nos travaux.
3. *Concernant la décentralisation* : L'architecture que nous proposons est par essence décentralisée, vu qu'un importateur peut adresser sa requête à n'importe quel médiateur de la fédération.
4. *Concernant la coopération* : C'est également une caractéristique de notre architecture vu que, quand un serveur local ne peut pas satisfaire une requête, il peut coopérer avec des partenaires, éventuellement hétérogènes, pour le calcul d'une réponse composée.

Sur un autre plan, considérant les caractéristiques des médiateurs (*gestion du contexte*, *propagation de requêtes*) introduites dans le chapitre 3 (section 3.1.3, page 63) :

1. Notre prototype n'offre pas de mécanismes sophistiqués de *gestion du contexte*. Il implémente des stratègies simples de détermination de réponses composées, comme *le premier qui satisfait une requête*, *au plus x* composants de la réponse, etc. Actuellement, pour supporter ces stratégies, un effort de programmation est requis : disposer d'un moteur générique pour supporter ces stratégies requiert des investigations supplémentaires.
2. En se basant sur cette gestion simplifiée du contexte, la *propagation de requêtes* est faite selon un critère de proximité (par opposition à un critère sémantqiue) : les requêtes sont toujours transmises au médiateur le plus proche comme une description de document en OWL-S.

De ce fait, la gestion d'une fédération dynamique, la gestion du contexte et la propagation de requête selon un critère sémantique reste encore à investiguer et à développer.

Néanmoins, ce prototype a le mérite de nous permettre de valider les théories et les propositions faites dans cette thèse même si d'autres extensions sont possibles, comme l'enrichissemnt du traducteur syntaxique entre langages de représentation, afin d'augementer la diversité de ces langages.

Toujours en termes de poursuites possibles de ce travail, nous pouvons envisager deux directions de recherche et d'applications : *(i)* extension de la théorie et de l'implémentation de la représentation de capacités et *(ii)* mise en œuvre dans le cadre d'applications réelles.

Tout d'abord, $\mathcal{ALN}_{r+}$ n'est pas suffisamment riche pour exprimer, par exemple, la négation etla disjonction (ce qui conduirait à quelques problèmes de NP-Complétude). Aussi, nous pensons explorer des langages de description qui supporteraient des descriptions plus complexes de concepts et de rôles ainsi que des algorithmes d'inférence sémantique complet. Un chemin possible vers ces objectifs, est de considérer les langages de représentation de connaissances fondés sur des graphes.

Sur un second plan, l'architecture de la médaiation fédérée se trouve confrontée à des problèmes de communication similaires à ceux rencontrés dans d'autres architectures hétérogènes distribuées comme celle des services Web du W3C, de CORBA de l'OMG ou de RM-ODP (*Reference Model for Open Distributed Processing*) de l'ISO. Dans cette thèse, nous avons défini n concept de description de capacités pour la communication de requêtes entre médiateurs. Pour des applications plus générales, il y a besoin d'une sorte de "langage de requête standard" (ou commun) qui permette de décrire de façon abstraite les requêtes de découvertes de capacités. Ce "standard" serait alors implémenté ou supporté par un ensemble de systèmes de représentation de connaissances.

3.7 Annexe : Application à la gestion de cartographie de connaissances

Alors que des *"exemples jouets"* sont utilisés dans les parties centrales de ce document, l'annexe contient un exemple partiel concret inspiré d'une expérience à laquelle nous avons participé pour la construction d'un système de gestion d'une carte de connaissances (*knowledge map*) [102].

Thèse (en anglais)

Chapitre 1

Introduction

The main motivation of this work is to provide a contribution to the satisfaction of the need for retrieving individuals (we will also call entities) who may carry out actions together with the retrieval of individuals who complementarily may carry out actions. In section 1.1, we introduce a definition for the term capability and we relate it to the concepts of competence and knowledge. We also intuitively define the notion of complementarity between capabilities and we will point out the needed concepts and mechanisms which contribute to the achievement of our goals. Section 1.2 summarizes our goals and our intended contributions, while section 1.3 outlines the structure and the content of this document.

1.1 Problem Statement : Capability, Competence and Knowledge

In the following, we call capability the ability to carry out actions. This definition is clearly very close to the definition of a competence and it is not independent from the notion of knowledge. Indeed, in these informal definitions, there is a very subtle and a very light distinction of the definition of what a capability is and the definition of a competence, in human resources, as being "the state or quality of being adequately or well qualified, having the ability to perform a specific role" [86]. However, all along the work which is reported here, we use the term capability in the above-mentioned meaning and we avoid, as long as possible, to use the term competence since this term is very often (implicitly) associated with the domain of human resource management. This avoidance is justified by the fact that we believe that our proposals can find their application in many domains, like competence management, Web services composition, enterprise knowledge management, e-business, etc.

Competence-based management originates from the theory of competence-based strategic management which is established as a theory in the early 1990's [103]. Some concepts on competence management are defined and compared in a certain hierarchy of assets in figure 1.1. The hierarchy of assets grants value to the separate assets in respect to each other. The higher it is in the hierarchy, the more complex the nature of the asset is. The capability and competence are two so related assets in the hierarchy. *"Capabilities are*

repeatable patterns of action in the use of assets to create, produce and/or offer products to a market. Because capabilities are intangible assets that determine the uses of tangible assets and other kinds of intangible assets, capabilities are considered to be an important special category of assets. Capabilities arise from the coordinated activities of groups of people who pool their individual skills in using assets to generate organizational action."

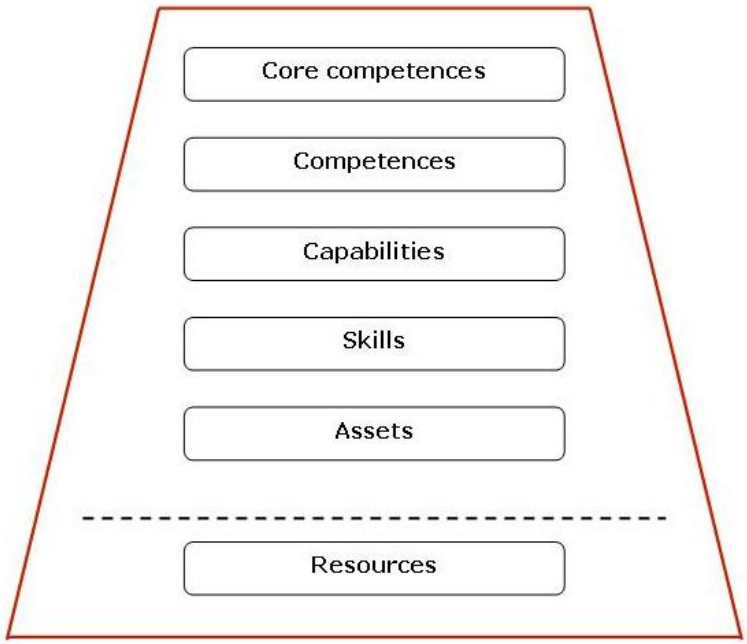


FIG. 1.1 – Hierarchy of Assets [103]

In [46], competence management means the definition, optimal use and development of competencies of the enterprises, the groups, and the individuals. A competence management system consists of four heavy processes : competence identification, competence assessment, competence acquisition, competence usage. The process of *competence identification* represents the competencies in a formal format which can be read in the processes of competence assessment and competence usage. The process of *competence assessment* fixes the relationship between individuals and required competencies, while the competencies of the individual satisfy the requirement.

A similar way, in the frame of enterprise knowledge management systems, [75] identifies various facets of enterprise knowledge, as summarized in figure 1.2. Enterprise knowledge (*explicit knowledge*) as well as enterprise Know-How (*tacit knowledge*) are essential for the decision processes and for the execution of the main processes which constitute the activity of an enterprise. Therefore, they need to be identified, located, characterized, organized into maps, evaluated and organized into hierarchies in order to serve for the enterprise purposes.

In *computer science*, the term capability was introduced by Dennis and Van Horn in 1966 in a paper entitled *Programming Semantics for Multiprogrammed Computations* [30]. The basic idea is the following : suppose we design a computer system in order to access an object, a program must have a special token. The token designates an object and gives

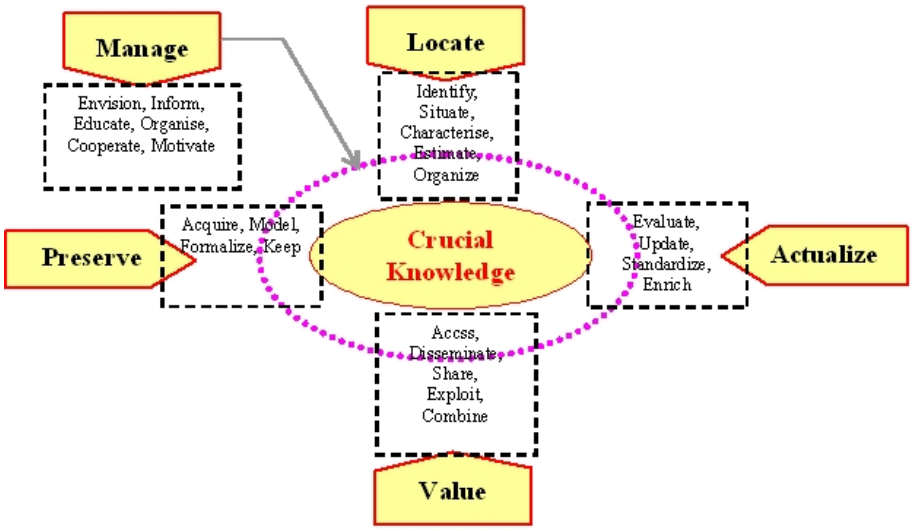


FIG. 1.2 – The Facets of Enterprise Knowledge Management

the program the authority to perform a specific set of actions (such as reading or writing) on that object. Such a token is known as a capability. We see that capability is provided by some intelligent agents in the domain of Artificial Intelligence (AI). These capabilities are essentially the actions an agent can perform solely or cooperatively [107].

All these above-mentioned considerations reveal the need, at least, for capability explicitation, capability discovery and capability composition.

Further, as said before, the (dynamic) *discovery* of the capabilities (or services) an "entity" offers has different application domains. Component-based programming, electronic business (*e-business*) and even enterprise knowledge management [75] are among the application domains in which there is a need for the discovery of services or capabilities an "entity" offers. For these purposes, the only syntactic description of an entity's capability (like the *signature* of a software component's service) is not satisfactory when using that description for answering a request : an additional semantic description is required. Moreover, the elicitation of possible relationships among the services may contribute to find out "the best" service or the "the best complementary" services that satisfy a search query. In *e-business*, considered as a possible application domain of this work, this notion of complementarity (similar to the notion of *pooling of individual skills* in [103]) can be applied when attempting to constitute business alliances or when looking for business partners. For example, when trying to constitute a business alliance, the notion of complementarity may help in retrieving the most appropriate candidates for partnership. The discovery of the complementarity can be performed when some explicitly-defined relationships among capabilities are known.

Indeed, there is a natural dependency structure among capabilities, that is, possessing a capability or a combination of capabilities normally entails possessing another. Brian R. Gaines[39] has extended the classification of capabilities as follows : *"It is tempting to extend this classification to the knowledge underlying the capabilities but this would be misleading since there is not a one-to-one relationship between knowledge and capabilities–*

usually, many different sets of knowledge can lead to the same capability".

This thesis work provides a formal background and a pragmatic solution to implement a competence management system which can describe, organize and discover competencies in heterogeneous environments. We use capability description to identify competencies, and the *competence assessment* can then be supported in a capability knowledge base. Capability discovery and matching are also valuable contributions to *competence acquisition and usage*, as defined in [46].

The goal and the overview of the thesis work is given in the coming section.

1.2 Thesis Overview and Thesis Contributions

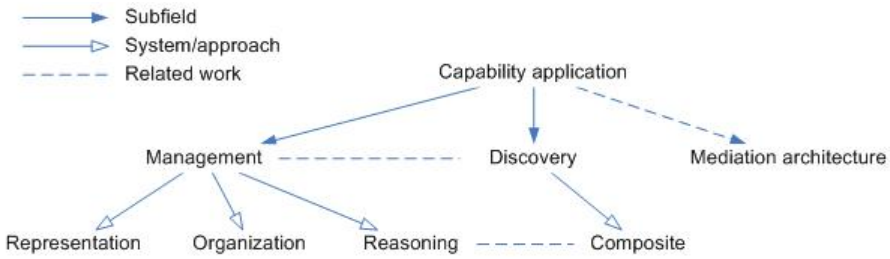


FIG. 1.3 – Top overview of capability application

A top overview of this work which provides two conceptualizations of the sub-fields capability management and capability discovery, and an implementation of the mediation architecture, is given in figure 1.3.

The *capability management* and the *capability discovery* are more like a pair of symbioses on capability application than two subfields of a capability application. It is a conceptual system model. This model is system-independent and it can be implemented on multiple system architectures. In our work, we have implemented this model on a mediated architecture which supports distributed heterogeneous federated mediators. We will identify these conceptual models and architectures in the next paragraphs.

Referring to the actions that are induced by the various facets of enterprise knowledge management (figure 1.2) the work that is reported in this document actually deals with :

1. the identification and the organization into hierarchies of capabilities owned by "entities", an entity being a physical or a non-physical individual,
2. the modeling and the formalization of these capabilities,
3. the access and the combination of the capabilities to satisfy a given goal expressed as a query.

Identification, organization, modeling and formalization are termed *capability management* and they are introduced in section 1.2.1, while capability access and combination are called *capability discovery* in section 1.2.2. Finally, section 1.2.3 introduces mediated architectures to support the search for complementarity between individuals.

1.2.1 Capability Management

There are two symbionts, *capability management* and *capability discovery*, in any capability application. We approach the capability management by two aspects, *capability representation* and *capability reasoning* such as showed in figure 1.3.

Knowledge Representation (KR) formalisms lend themselves to *capability representation*. The field of knowledge representation has, of course, long been a focal point of research in the AI community [90]. These current KR languages are not totally competent for capability representation. Capability is just an attribute of an entity in many KR languages, where we can not describe the characteristic of the capability itself. For example, we can see that the capability description of "be brother of" is transitive, and the pair of capability "be father of" and "be son of" is symmetric. These characteristics of capability and relationships between capabilities are hard to describe in current KR languages. Thus, we need extended current KR language to be competent for capability representation.

For capability management, of course, the aim of representing capabilities is that we want to reason on them. Hence we do not only need this formula knowledge representation language for capability description. This capability representation language must own strong capability description features, and it must support some basic logical reasoning rules. The capability matching is one of the most interesting feature in the reasoning tasks in our work ; it will be the basis of the capability discovery.

In addition, from a knowledge representation perspective and regardless of any actual knowledge description language, the role of verbs [55] helps to accurately discriminate documents by types and semantic properties, rather than nouns. Parallel problems are also argued in knowledge representation field, such as the two kinds of description logic language, relation-based and concept-based, that are used in natural language processing [38], and relation-based description logic can be embedded in concept-based description logic. The $\mathcal{ALN}_{r+}$ description logics language is a kind of this embedded language, but it develops the relationship description. We will rely on it for formalizing capabilities and some of their semantic relationships. The formal approaches based on description logics formalization and reasoning, are applied, among others, in semantic matching [28] and Semantic Web composition [79]. *The capability discovery approach of our prototype implements semantic matching and service composition in a heterogeneous environment. Therefore, this thesis works is also a valuable contribution to semantic technical applications*, like semantic-based discovery of Web services [106] and it is in line with current activities on semantic and ontology for the Web [99, 36, 105, 104, 1].

1.2.2 Capability Discovery

Here capability discovery is defined in a query-answer context. Intuitively saying, a capability discovery action tries to find some capabilities to satisfy a given query. Capability discovery is based on a semantic query model such as the general business discovery model in figure 1.4.

"Business need" is described in a semantic query language. It gets two results in two different levels, "business solution" and "service components". Where "business solution" is a conceptual description, it does not point one/multiple companies or business entities,

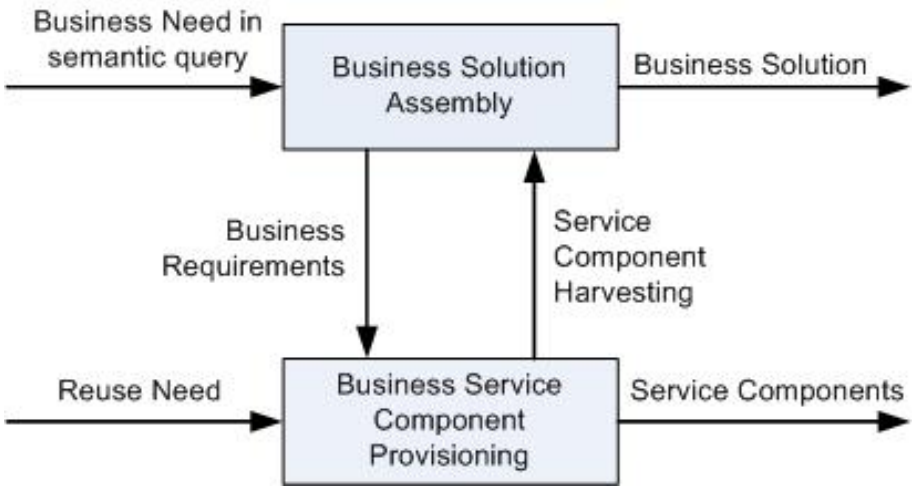


FIG. 1.4 – Business Discovery Model

and it describes which kind of companies or business entities can be in the "business solution". And then we also can get a list of individuals, "service components". It directly points out which company or business entity can satisfy the "business need". "Reuse need" is just a normal entity-relation search here.

Therefore, *capability discovery* also works like this semantic query model. It tries to find which kind of entities owns a capability, and who they are. This *query action* is working on a knowledge base that is written in a KR language. In many situations, the knowledge bases are possibly distributed or heterogeneous, as in an internet environment.

In this work, we approach the *capability discovery at* three levels :

1. Firstly, the query and the knowledge base are in the same KR language, and then the knowledge base is located in one server. This level focuses on the matching calculation and composite answer calculation. These are the most basic algorithms and approaches. We call this level the *homogeneous local satisfaction.*
2. Secondly, the query and the knowledge base are in the same KR language and the knowledge base is distributed in several independent servers. To support, these calculus and approaches of *homogeneous local satisfaction* work in distributed environment. This second situation focuses on the solution of distributed calculation problem. We call this level the *homogeneous distributed satisfaction.*
3. Finally, the third level is called *heterogeneous satisfaction.* The query and the knowledge base are in different KR languages, and the knowledge base may be distributed. This situation focuses on the problem of communication and interoperation between different KR languages.

The three satisfaction levels mutually benefit one from the other, and they also focus on different dedicated fields. The homogeneous distributed satisfaction and heterogeneous satisfaction can together be applied to many current applications. That means that capability discovery can work in distributed heterogeneous environments.

More precisely, among central objectives and original contributions of our work, we intend to propose :

1. a formal framework for describing and organizing capabilities : we emphasize the need for an appropriate formal and sound basis for the representation and the organization of the capabilities we intend to describe and manage. From our concern, we opted for the description logic family of languages.
2. a single concept and some associated mechanisms that fit the three previous levels and that enable :
   (a) comparing intentionally defined entities (i.e. entities described by a set of capabilities),
   (b) identifying capability gaps between given intentionally defined entities,
   (c) identifying entities that are candidates for filling-up identified gaps,
   (d) constraining the set of candidates that contribute to the fulfillment of identified gaps.

   The fundamental concept which underlies this work is the *complement concept*, i.e. the identification of *"what is missing to an entity to be considered as an entity that meets a given set of capabilities"*. This concept is very central in this work since it provides the accurate basis, at the same time, for *(a)* comparing intentionally defined entities, *(b)* identifying the possible gaps *(c)* full filling these gaps. The algorithm we developed and that is detailed in the chapter 5 covers all these situations.

These contributions are validated thanks to a mediation architecture we have designed and (partly) implemented as a proof of concept for capability management applications.

1.2.3 Mediation Architecture

Mediation architecture is an active research field in network program design. A central role is assigned to modules that mediate between the users' workstation and data resources. Mediators are defined as modules occupying an explicit, active layer between the users' application and the data resources [45]. Thousands of different kinds of mediation services are working in different specific fields right now. These mediators serve in different specific fields in different physical networks, and also use different KR languages and different communication interfaces. We need a federation of mediators when a complex application wants to access data on multiple mediators on different specific fields. The federation can make cooperate multiple independent mediator servers, translate KR languages between these mediators, and integrate the results into a unique homogeneous answer.

In this work, mediator-based architectures are explored to meet the facilities that are needed by *homogeneous distributed satisfaction* as well as by *heterogeneous satisfaction* of capability discovery. In the first situation, only communication is needed between some mediators, as long as the service that is needed from them is identified : the needed service is the identification of the *complementary entities*. In the second situation (*heterogeneous satisfaction*), an additional functionality is needed, in addition to the communication one : it is the functionality that helps two heterogeneous systems to unambiguously understand each other. For this purpose, clearly the choice of a common vocabulary and the availability of translation mechanisms will greatly help in the communication between different

capability management systems. We pragmatically approached this problem thanks the inclusion in the mediation architecture of term translators.

1.3 Thesis outline

The thesis is organized as follows :

Chapter 2 provides an introduction on the state of art for the related research areas involved in the thesis work. There are three main related research areas : *Knowledge Representation*, *Capability Discovery*, and *Mediation Architecture*. We observe the different characteristics on *Capability Representation* between some knowledge representation languages, and then we mention the rewriting problems on the *Capability Representations* in different knowledge representation languages. The *Capability Discovery* is the approach of *Semantic Matching* which is related with multiple research fields : *Lexical Integration*, *Schema Matching*, *Web semantic*, etc... The *Mediation Architecture* is a hybrid, *Distributed, Heterogeneous, and Dynamic* system architecture. It is related with multiple *Distributed System Architectures* : *Peer to Peer*, *Web services*, and so on.

Basically, by introducing findings from literature and from empirical studies in research projects, chapter 2 describes capability representation theories and capability discovery approaches and mediation architectures.

Chapter 3 presents the overview of the approach and the system architecture. We go through a conceptual model of federated mediators, composite answers and knowledge base management. That accords to three main research lines :

1) the study of knowledge representation languages devoted to capability and *concept* description ;
2) the description of our approaches (e.g. composite answer, complement calculation, classification) ;
3) the study and the design of an heterogeneous and distributed federation of mediators.

This chapter sketches the overview of our proposals and the three next chapters detail the three considered research lines.

Chapter 4 focuses on the application of the knowledge representation theories to *capability representation*. It proposes a capability description language $\mathcal{ALN}_{r+}$ which extends the Description Logic's language $\mathcal{ALN}$. We introduce some new syntax restrictions in $\mathcal{ALN}_{r+}$ for capability description. And then we introduce $\mathcal{ALN}_{r+}$ into a Description Logics knowledge management system. The capability management model in $\mathcal{ALN}_{r+}$ develops from the classical DLs-system model which can further be divided into the TBox (conceptual knowledge) and the ABox (assertion knowledge). $\mathcal{ALN}_{r+}$ allows an extended syntax for capability description, and then $\mathcal{ALN}_{r+}$-system further divides T_rBox from TBox. A T_rBox specifically stores the capability knowledge (the role vocabulary) in the TBox of an application domain.

In short, we introduce some new capability description syntaxes and a capability description space into classical knowledge representation systems.

Chapter 5 describes theories and algorithms for logical reasoning on $\mathcal{ALN}_{r+}$ knowledge representation systems. The *satisfiable* and *subsume* are the most important inferences and where we will introduce our approach and algorithms. The inferences are implemented using a Normalization-Comparison algorithm for testing the existence of the subsumption relationships between two description structures which could be some entity description (i.e. considering the ABox and reasoning using its content of) or some capability description (i.e. considering the T_rBox and reasoning using its content). The Normalization-Comparison algorithm proceeds in two phases : first, the description structures are normalized into a normal description formula, and then the normal description structures are compared to implement the inference services.

In addition, chapter 5 introduces the approach and the strategy for *capability discovery*. The approach for *capability discovery* is based on a model of *composite answers*. A *composite answer* is not a "Yes/No" response to a query, but it may contain two components : a satisfaction part and a complement part. The satisfaction component is the part of the query that is satisfied by the mediator to which the initial query is addressed, i.e. it designates the individuals whose capabilities cover part of those required in the query. The complement component is the part that is missing in the satisfaction part to cover the whole query. The satisfaction part is determined thanks to the subsumption relationship, while the complementary part is determined thanks to a procedural implementation of the *complement concept* [91].

Chapter 6 reports on the capability application's approaches implementation in a federated mediators prototype. We show some details of the implementation of the prototype system on the two parts. Firstly, we will introduce the implementation of the mediator server in $\mathcal{ALN}_{r+}$, which has been implemented in *Java*. Secondly, we focus on the implementation of the federation of mediator servers, which is the heterogeneous environment. The *Web services* technologies serve the composition between the heterogeneous mediator serves.

Chapter 7 summarizes the results of this thesis and concludes with an outlook on further work.

Appendix : While *"toy examples"* are used in the core of this document, the appendix contains a partial concrete example inspired from an experiment we had in building a knowledge map management system [102].

Chapitre 2

Knowledge Management and Distributed Heterogeneous Architectures

This chapter provides an introduction on related works for the main research areas involved in the thesis work. An overview of this chapter, which provides a conceptualization of the relationships between the different sub-fields and approaches/systems described in this thesis is given in figure 2.1.

In section 2.1 we introduce the basics of knowledge representation theories and languages. Both concepts are fundamental for capability representation. These approaches of capability management and discovery are independent of knowledge representation. There are two meanings of "independence" : *(i)* the approaches can be implemented on any platform and use any knowledge representation technology and *(ii)* the approaches allow heterogeneous interoperable systems to use different knowledge representation technologies. Concretely, we will illustrate our work using three common knowledge representation technologies : *Description Logics*, *Frame Logics* and *Conceptual Graphs*. We will define some basic terms which are accepted by these knowledge representation technologies. We will use Description Logics to represent our approach in this thesis work, so we will move the focus on Description Logics in section 2.1.1. We will compare several language families of DLs in section 2.1.1. We will provide extensive examples to ease the understanding of knowledge representation in DLs. In section 2.1.2 we introduce theories and languages on Frame Logics, and related projects. In section 2.1.3 we will introduce theories and languages on Conceptual Graph, and related projects.

In section 2.2 we introduce theories and approaches that are related to capability discovery. Knowledge integration and semantic matching are two core problems when we are in an heterogeneous software environment. We will focus on mapping on lexical level in section 2.2.1, where there are some ontological theories and related projection. Semantic matching focuses on some few concepts and calculations of matching on the syntax level in section 2.2.2.

In section 2.3 we introduce distributed heterogeneous architectures. As mentioned in section 1.2.3, we will like to call Mediation architecture, which is a heterogeneous and distributed federated mediators in this thesis work. Therefore, we introduce some basic

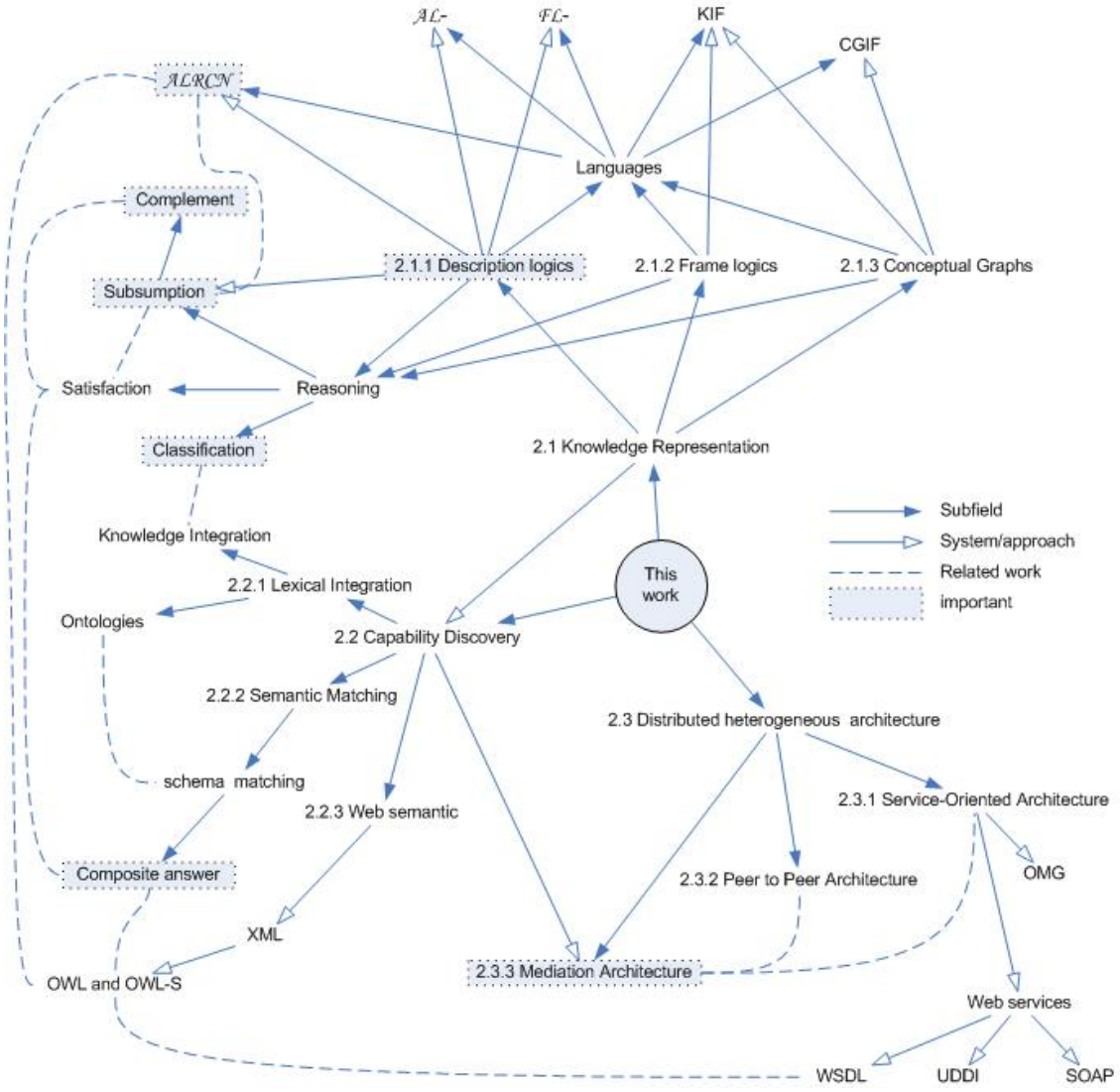


FIG. 2.1 – The Knowledge Map of the thesis work outlines this chapter.

concepts on mediation architecture in section 2.3. Heterogeneous and distributed systems are a long-term research focus. Thus, there exist numerous heterogeneous and distributed systems architectures. We devise these architectures into three types, Service-Oriented Architecture, Peer to Peer, and Mediation. We introduce Service-Oriented architecture where we focus on the most popular SOA architecture, Web service architecture in section 2.3.1. And then we will mention the other common OMG architecture which is based on CORBA. We argue the three distributed architectures by knowledge management application on Peer to Peer in section 2.3.2. In our view, mediation architecture is a hybrid distributed software architecture. We introduce their basic concepts in section 2.3.3.

2.1 Knowledge representation

In this section we look at how some logics have been or could be used in capability representation. The linguistic representation plays a special role for the usefulness of heterogeneous reasoning systems [9]. Firstly, in linguistic representation, it is very useful to have names for things that have different representations in different systems. These names are frequently letters or words. Secondly, reasoning problems often come to us posed wholly or partly in a language, and the results of our reasoning typically need to be expressed in a language. Thus, some linguistic problems on capability representation, which are argued in knowledge representation domain, firstly move in our view in this thesis work.

Capability representation is one of the sub-fields of knowledge representation which extends current knowledge representation languages to be more competent for capability description. Some basic theories and concepts of knowledge representation, that we will introduce in the following paragraph, are primarily for well talking in this thesis.

Knowledge Representation (KR) has been one of the focal research fields and has a long history in Artificial Intelligence (AI) domain. What is KR ? That is the most fundamental question, which has rarely been answered directly. Randall Davis (1993) [89] uses distinctly five different roles to describe what is KR.

– "*A KR is a Surrogate.*"
– "*A KR is a Set of Ontological Commitments.*"
– "*A KR is a Fragmentary Theory of Intelligent Reasoning.*"
– "*A KR is a Medium for Efficient Computation.*"
– "*A KR is a Medium of Human Expression.*"

In one word, a KR is an intelligent medium which is working among AI systems and humans. Almost every KR theory is based on mathematical set theory where elements, sets and relationships are the three most basic concepts [97].

In many applications, the elements are never defined, but are left as abstraction that could be represented in many different ways in the human brain, on a piece of paper or in computer storage. In this thesis, the elements are descriptions of capability provider which include all location, binding and ports information in service description languages. But they only exist as identifies which we usually name *term* in KR languages.

A set is an arbitrary collection of elements, which may be real or imaginary, physical

or abstract. In mathematics, sets are usually composed of abstract things like numbers and points, but one can also talk about sets of acts, roles, peoples, or a service whose elements have a kind of capability or characteristic.

A relation is a function of one or more arguments whose range is the set of truth values true, false. Some KR systems deem that relations must have two or more arguments and call a relation with one argument a property. For example, "roles are imaginary" that is a unitary relation, where "are imaginary" is a property of "roles". And then "acts play roles" is a binary relation, where "play" is a relation between "acts" and "roles".

Generic knowledge representation formalisms such as first-order predicate logic (FOPL) might have turned out to be sufficient for KR. FOPL provides very powerful and general machinery ; logic-based approaches were more general-purpose from the very start. Advantages of FOPL include its well-defined semantics and the fact that it is probably the best-researched knowledge representation formalism in AI. In a FOPL approach, the representation language is usually a variant of the first-order predicate calculus, and reasoning amounts to verifying logical consequences. Thus, many current KR systems agree on the FOPL, such as Description Logics, Frame-Logics and controlled natural languages. We will successively introduce these KR theories and systems in the following sections.

2.1.1 Description Logics

Description Logics (DLs) [32, 71, 6] owns several families of knowledge representation languages. The earliest Description Logic (DL) pre-system derived directly from KL-ONE [24], while this system is a direct result of a formal analysis of the shortcomings of semantic networks. KL-ONE is mainly about the implementation of a viable classification algorithm and the data structures to adequately represent concepts.

The current DL knowledge representation system has complete algorithms for DL languages. The expressiveness of the DL language required for reasoning on data models and semi-structured data has contributed to the identification of the most important extensions for practical application. A KR system based on DLs provides facilities to set up knowledge bases, to reason about their content and to manipulate them. Figure 2.2 sketches the architecture of such a system.

In this Knowledge Base (KB) architecture, there are two components : the Terminology Box ($TBox$) and the Assertions Box ($ABox$). The $TBox$ introduces the terminologies of the modeled world and it includes the names of the *concepts* and the names of the *roles*, while the $ABox$ contains assertions about *individuals* in terms of the vocabulary. Therefore, a DL knowledge base not only stores terminologies and assertions, but it also offers some services for reasoning about them. So we introduce hereafter some typical KR languages and reasonings starting from preliminary definitions.

Preliminary Definitions and Example

In DLs knowledge representation languages, a description of a world is built using *concepts*, *roles* and *individuals*. The *concepts* model classes (sets of *concepts*, TBox) of individuals (sets of *individuals*, ABox) and they correspond to generic entities in an application domain. An *individual* is an instance of a *concept*. *Roles* model binary relationships

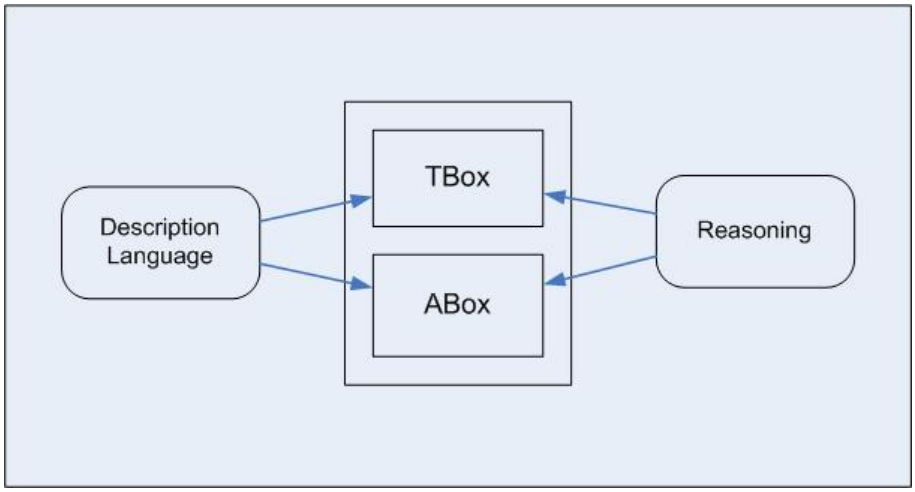


FIG. 2.2 – Architecture of knowledge representation system based on DLs

among the individual classes. A *concept* is specified thanks to a structured description that is built giving *constructors*. It introduces the *roles* associated with the *concept* and possible restrictions associated with some roles. Usually, the restrictions constrain the range of a binary relationship that is itself defined by a role and related cardinalities.

Concepts are of two types : primitive and defined concepts. *Primitive concepts* may be considered as atoms that may serve to build new *concepts* (the *defined concepts*). Similarly, *roles* may be primitive *roles* as well as defined roles. In the figure 2.3, PERSON and SET are primitive *concepts* : they are introduced using the symbol $\dot{\sqsubseteq}$ and they are linked to a "TOP" *concept* $(\top)$[2] ; TEAM SMALL-TEAM and MODERN-TEAM are defined *concepts* (they are introduced using the symbol $\doteq$). The *and* constructor enables defining *concepts* as a conjunction of *concepts* : these *concepts* are the immediate ascendants of the defined one. The *all* constructor constrains a role's range and the *at-least* and *at-most* constructors enable to specify the role's cardinals. Finally, the *not* constructor only applies to primitive *concept*.

Let us now turn our attention toward two significant languages, $\mathcal{FL}$ and $\mathcal{AL}$ languages successively.

$\mathcal{FL}$ Languages

$\mathcal{FL}^-$ language is probably the first DL language. In the paper "The tractability of subsumption in frame-base description languages", Brachman and Levesque [23] argued that there is a tradeoff between the expressiveness of a representation language and the difficulty of reasoning over the representation built using that language. They provided the first example of this tradeoff by analyzing the language $\mathcal{FL}^-$ (Frame Language), which included intersection of *concepts*, value restrictions and a simple form of existential quantification. This paper introduced two new ideas :

[2]Intuitively the TOP *concept* is the "most general one" and it contains all the individuals while the BOTTOM *concept* $(\bot)$ is the most specific one and it is empty.

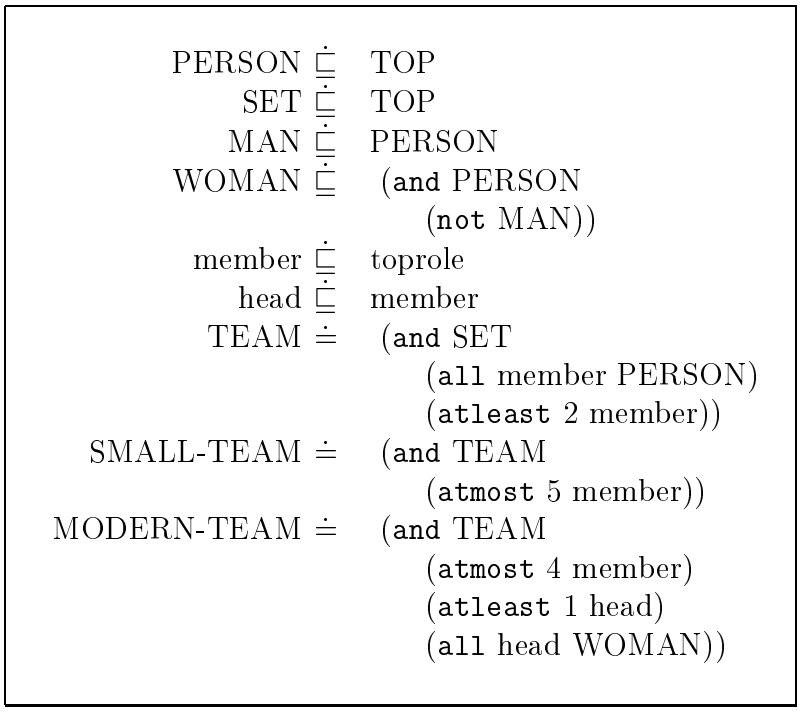


FIG. 2.3 – A collection of primitive *concepts*, primitive *roles*, and defined *concepts* in [73]

1. "efficiency of reasoning" over knowledge structures can be studied using the tools of computational complexity theory ;
2. and different combinations of constructs can give rise to languages with different computational properties.

$\mathcal{FL}$ is a significant extension of $\mathcal{FL}^-$. In the following, we use the letters A and B for primitive *concepts*[3], the letter p for primitive roles, the letter r for roles, and the letters C and D for *concept descriptions*. Description languages are distinguished by the constructors they provide.

n $\mathcal{FL}$, *concept* and *restrictions description* are formed according to the following syntax rules :

C, D	$\longrightarrow$	$A \mid (\texttt{and}\ C\ D) \mid (\texttt{all}\ r, C) \mid (\texttt{some}\ r)$
r	$\longrightarrow$	$p \mid (\texttt{restrict}\ r\ C)$

TAB. 2.1 – The syntax of $\mathcal{FL}$

This syntax is inherited by subsequent description languages, like $\mathcal{AL}$ which is introduced below. Exceptionality, the `restrict` structure, reaches the *roles* description of $\mathcal{FL}$. It introduces a restriction to the co-domain of a role. The following example (in [71])

[3]Note that some use the notion "atomic concept" to the primitive *concept*, where "atomic concept" specially names the minimal *concept* after the Normalization algorithm in this thesis work.

shows part of what can be expressed in $\mathcal{FL}$: the *concept* D represents "One person has children and all poor children are musicians while all rich children are doctors" :

$D \doteq$ (`and` PERSON
(`some` child)
(`all` (restrict child POOR) MUSICIAN)
`all` (restrict child RICH) DOCTOR))

The $\mathcal{FL}-$ language is a simplification of $\mathcal{FL}$ without the role restrict structure.

$\mathcal{AL}$ Languages

The $\mathcal{AL}$ language (Attributive Language) has been introduced by Schmidt Schauß and Smolka in 1991 [92] as a minimal language that is of practical interest. The other languages of this family are extensions of $\mathcal{AL}$. *Concept* and restrictions description in $\mathcal{AL}$ are formed according to the following syntax rule :

$C, D \longrightarrow$	A |	(primitive concept)
	$\top$ |	(universal concept)
	$\bot$ |	(bottom concept)
	$C \sqcap D$ |	(intersection)
	$\neg A$ |	(primitive negation)
	$\forall\ r.\ C$ |	(value restriction)
	$\exists\ r.\top$	(limited existential quantification)

TAB. 2.2 – The syntax of $\mathcal{AL}$

In $\mathcal{AL}$, negation can only be applied to primitive *concepts*, and only the universal concept is allowed in the scope of an existential quantification over a role. As mentioned in the previous section, the $\mathcal{FL}^-$ can be seen as the sub-language of $\mathcal{AL}$, that disallows primitive negation. Contrarily, adding a new constructors to $\mathcal{AL}$ results into a new language. For example, $\mathcal{ALC}$ is obtained by the addition of the negation constructor to the defined concept in $\mathcal{AL}$: $\mathcal{ALC} = \mathcal{AL} \cup \{\neg C\}$, where The negation of primitive or defined *concepts* is written $\neg C$. This extension of $\mathcal{AL}$ is indicated by $\mathcal{ALC}$ where the letter $\mathcal{C}$ stands for "complement".

The new language $\mathcal{ALC}$ supports all description constructors in $\mathcal{AL}$, and it allows the negation operation over any concept descriptions which include the primitive concept and defined concept. Knowing that $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$, then the $\mathcal{ALC}$ language can describe the union of concepts. Therefore, $\mathcal{ALC}$ is more expressive language than $\mathcal{AL}$. We may add further constructors to $\mathcal{AL}$ to obtain a family of $\mathcal{AL}$-languages.

- $\mathcal{AL}\mathcal{U} = \mathcal{AL} \cup \{C \sqcup D\}$: The union of *concepts* is written $C \sqcup D$. Note that equivalences are $\bot \equiv C \sqcup \neg C$ and $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$ [92].
- $\mathcal{AL}\mathcal{E} = \mathcal{AL} \cup \{\exists r.C\}$: Full existential quantification is written $\exists r.C$. Note that $\exists r.C$ differs from $\exists r.\bot$ in that arbitrary *concepts* are allowed to occur in the scope of the existential quantifier, and then that equivalences are $\exists r \equiv \exists r.\bot$ and $\exists r.C \equiv \neg(\forall r.\neg C)$ [35].
- $\mathcal{AL}\mathcal{N} = \mathcal{AL} \cup \{\leq n\ r, \geq n\ r\}$: Number restrictions are written $\leq n\ r$ (at-most restriction) and $\geq n\ r$ (at-least restriction), where n ranges over the nonnegative integers.
- $\mathcal{AL}\mathcal{R} = \mathcal{AL} \cup \{r_1 \sqcap r_2\}$: The intersection of *roles* is written $\{r_1 \sqcap r_2\}$, where r_1 and r_2 are primitive roles. If $r = r_1 \sqcap r_2$, then r is a sub role from r_1 and r_2. It is possible to construct a hierarchy of roles.

With the additional constructors, we can, for example, describe those persons that have not more than three children, one of whom is rich.

$D \doteq$ (`and` PERSON
($\leq$ 3 child)
`some` (restrict child RICH)))

Extending $\mathcal{AL}$ by any subset of the above constructors yields a particular $\mathcal{AL}$-language. We can name each $\mathcal{AL}$-language by a string of the form

$$\mathcal{AL}[\mathcal{C}][\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{R}]$$

where a letter in the name stands for the presence of the corresponding constructor.

Following the equivalences mentioned above is not all these languages are distinct from the semantic point of view. The semantics enforces the equivalences $C \sqcup D \equiv \neg(\neg C \sqcap \neg D$ and $\exists R.C \equiv \neg\forall R.\neg C$. Hence, all $\mathcal{AL}$-languages can be written using the letters $\mathcal{C}, \mathcal{N}, \mathcal{R}$ only, that is the rich semantic language $\mathcal{ALCNR}$.

Subsumption Relationship

The *subsumption*, denoted as $C \sqsubseteq D$, is the most basic inference on *concept* expressions in DL. The *subsumed concept* C, always denotes a subset of the *subsumer concept* D, as for instance `MAN` $\sqsubseteq$ `PERSON`. The *subsumption* relation is reflexive (one *concept* is subsumed by itself), transitive (if E is subsumed by D and D is subsumed by C, E is subsumed by C), and antisymmetric (if D is subsumed by C and C is subsumed by D, $C = D$). DLs has been influenced for a longtime by the tradition of semantic networks in their history, where the *concepts* were viewed as nodes and *roles* as links in a network. *Subsumption* relation reorganizes the *concepts* and roles in hierarchy. This hierarchy owns one maximal

element, `TOP`, which subsumes all other elements, and one minimal element, `BOTTOM`, which is subsumed by all other elements.

Traditionally, the basic reasoning mechanism provided by DL systems checked the subsumption of concepts. In fact, *subsumption* can draw other important inferences, as shown by the following reduction. For two *concepts* C, D we have :

Satisfiability C is not satisfiable $\Leftrightarrow$ C is subsumed by $\bot$;

Equivalence C and D are equivalent $\Leftrightarrow$ C is subsumed by D and D is subsumed by C ;

Disjointness C and D are disjoint $\Leftrightarrow$ $C \sqcap D$ is subsumed by $\bot$;

Thus, checking subsumption is a key inference in Description Logic systems. The earliest subsumption algorithms was to transform two input *concepts* into labeled graphs and test whether one could be embedded into the other ; the embedded graph world correspond to the more general *concept* (the subsumer) [59]. Until now, there are two principal subsumption checking algorithms : the algorithms of normalization-comparison types (or call *NC algorithms*) and tableau-based algorithms [47]. The description languages of all the early and also of some of the present day DL systems do not allow negation. For such Description Logics, subsumption of *concepts* can usually be computed by *NC algorithms*. In particular, Description Logics with negation and disjunction can not be handled by *NC algorithms*. For such language, tableau-based algorithms have turned out to be very useful. The first tableau-based algorithms was introduced by [92].

Since 1975, Brachman started to develop KL-ONE [21] [22], a lot of DL systems have been implemented. The first systems include KL-ONE, KRYPTON [24]. Then, successor systems are described by classifying them along the characteristics discussed in the previous sections, addressing the following systems : CLASSIC [16] [58], BACK [87] [83] [50], LOOM [60] [61]. Finally, a new optimized generation of very expressive but sound and complete DL systems, such as FACT [52], and RACER [69].

In addition to Description Logics, we considered Frame Logics and Conceptual Graphs to study and experiment cooperation among heterogeneous systems. These are briefly introduced in the two next sections (sections 2.1.2 and 2.1.3, respectively) and they are compared in section 2.1.4.

2.1.2 Frame Logics

Michael Kifer, Georg Lausen and James Wu introduce Frame Logic (F-logic) [54], that accounts in a clean and declarative fashion for most structural aspects of object-oriented and frame-based languages. It borrows and turns ideas from O-logic [53]. F-logic is a full-fledged logic ; it has a model-theoretic semantics, a sound and complete proof theory.

F-logic organizes the classes and individual objects in an IS-A hierarchy. Figure 2.4 shows part of a hierarchy of classes and individual objects which is introduced in [54], where solid arcs represent the subclass relationship and dotted arcs represent class memberships. This hierarchy asserts that *faculty* and *manager* are subclasses of *empl* ; "Bob" and "Mary" are members of the class *string* ; and *mary* is a *faculty*. This endows F-logic

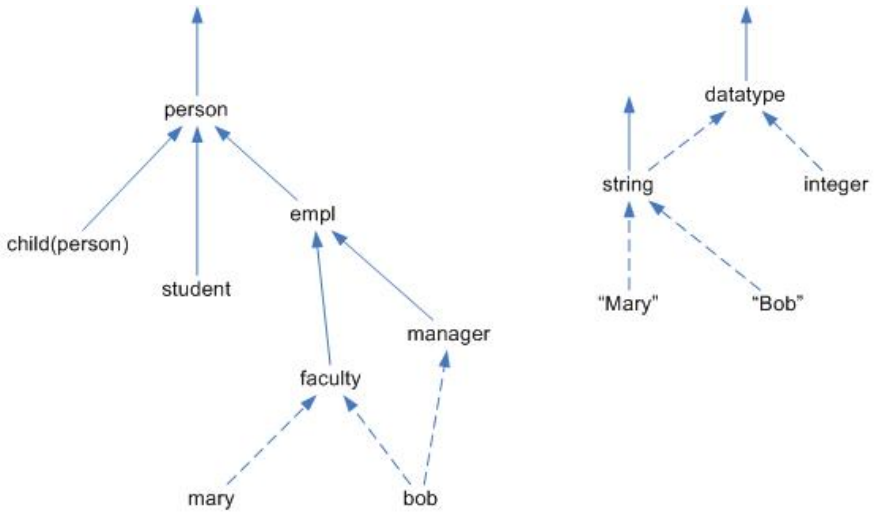


FIG. 2.4 – Part of an IS-A hierarchy in [54]

with a great deal of uniformity, making it possible to manipulate classes and objects in the same language.

In the actual syntax of F-logic, " :" is used to represent class membership and " : :" to denote the subclass-relationship. Thus, for instance, the expression in the hierarchy of figure 2.4 can be recorded as $empl :: person$, $bob : manager$, and so on. F-logic allows more symbols for objects, classes and deductive rules description. In short, $\rightarrow$ and $\Rightarrow$ signify scalar attributes, $\rightarrow\rightarrow$ and $\Rightarrow\Rightarrow$ are used with set-valued attributes. Also, double-shafted specify types, while arrows describe values of attributes. For example,

Database facts :

bob [$name \rightarrow$ "Bob" ; %Defining a scalar property name of bob
$age \rightarrow 40$; %Defining a scalar property age of bob
$friends \rightarrow\rightarrow \{mary, sally\}$]

General class information :

$person$ [$name \Rightarrow String$; %Attribute $name$ returns object string
$age \Rightarrow midaged$; %Attribute age returning object from class $midaged$
$friends \Rightarrow\Rightarrow person$] %Returns sets of objects from class $person$

Deductive rules :

$$E[boss \rightarrow M] \leftarrow E : empl \wedge D : dept \wedge E[affiliation \rightarrow D[mngr \rightarrow M : emp;]]$$

Queries :

$$? - X : person \wedge X[\, name \rightarrow Y; \; age \rightarrow Z : midaged; \; friends \rightarrow\rightarrow D[name \rightarrow "Mary"]]$$

A whole database consists of four parts : *database facts*, *general class information*, *deductive rules* and *queries*. *database facts* defines the scalar properties of objects, that are descriptions of individuals. It can correspond to the $ABox$ in Description Logics.

general class information describes the information of classes, which is like the definition of *concepts* in a *TBox*. Finally, *deductive rules* are descriptions on the relations between classes, which correspond to description of roles in a Description Logics' *TBox*.

2.1.3 Conceptual Graphs

Conceptual graphs (CGs) are a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence. They express meaning in a form that is logically precise, humanly readable and computationally tractable. With their direct mapping to a language, conceptual graphs serve as an intermediate language for translating computer-oriented formalisms to and from natural languages. With their graphic representation, they serve as a readable, but formal design and specification language. CGs have been implemented in a variety of projects for information retrieval, database design, expert systems, and natural language processing.

The default quantifier in a concept is the existential $\exists$, which is normally represented by a blank. For example, in figure 2.5, the *concept* [Cat] without anything in the referent field is logically equivalent to the *concept* [Cat : $\exists$], which asserts the proposition according to which there exists a cat. Other quantifiers, such as the universal $\forall$, are called defined quantifiers because they can be defined in terms of conceptual graphs containing only the default existential. In figure 2.5, the *concept* [Cat : $\forall$] represents the phrase "every cat", and the complete CG represents the sentence "Every cat is on a mat".

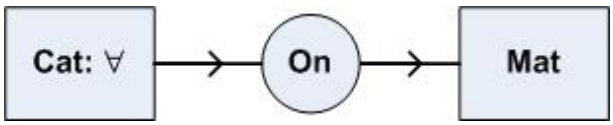


FIG. 2.5 – An example "*Every cat is on a mat.*" on Conceptual Graph

There are multiple candidate representations for CGs [64]. In the CG linear form (CGLF), the universal quantifier may be represented by the symbol @every [95] [96]. CG Interchange Format (CGIF) is intended to be one of the standard notations for exchanging knowledge, and it is the official textual notation for CGs. Knowledge Interchange Format (KIF) is a language designed to be used in the interchange of knowledge among disparate computer systems [42] [44]. As an example, the following CGs descriptions are semantically identical :

CGLF : [Cat : @every]→(On)→[Mat].

CGIF : [Cat : @every*x] [Mat : *y] (On ?x ?y)

KIF : (forall ((?x Cat)) (exists ((?y Mat)) (On ?x ?y)))

2.1.4 Concluding Remarks

We already mentioned some advantages of capability discovery approach in heterogeneous environments. We introduced three common types of knowledge representation

technology in section 2.1 and their concepts are compared in the table 2.3.

	DLs	**F-logics**	**CGs**
Concept Description	Concept	Class	Concept,
Capability Description	Role	Class : :Method	Relation
Individual Description	assertion	object	individual
Knowledge Organization	subsumption hierarchy	IS-A hierarchy	directed graph
Reasoning method	structural comparison, tableau method	sorted F-logic, multiset-valued method	inverting resolution,

TAB. 2.3 – Knowledge Representation Technologies comparison

From the table 2.3, it comes that the studied Knowledge Representation formalisms allow concept level and individual level descriptions. Sorted F-logic and multiset-valued method were introduced in [54] and they are common methods of F-logics extensions. Methods for performing inductive inference is an important and useful part of AI. It has been applied to CGs reasoning [77]. In this thesis work, we mainly focus on the *capability description* in DL, together with the *capability description* in multiple knowledge description languages to illustrate the capability discovery in heterogeneous context. Heterogeneity usually introduces additional problems like knowledge translation from a representation to an other one or knowledge integration. In the coming section, we briefly present some notions about integration which we used when studying capability discovery in a heterogeneous environment.

2.2 Capability discovery

This capability discovery approach is mainly related to three current research fields : *Knowledge Integration*, *Semantic Matching*, and *Semantic Web*.

Knowledge Integration looks so much like the ontology work, which recently became a very popular research domain in AI. *An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence.* This definition is given in [48]. An informal ontology may be specified by a catalog of types that are either undefined or defined only by statements in a natural language. A formal ontology is specified by a collection of names for concept and relation types organized in a partial ordering by the type-subtype relationship (Definitions and concepts relative to the ontology that are presented in this thesis are based on the book Knowledge Representation by John F. Sowa [97]). The basic categories and distinctions have been derived from a variety of sources in logic, linguistics, philosophy, and artificial intelligence. *However, in this thesis work, the ontology domain is not a main concern. So, we will only consider some lexical applications of ontologies, that support terms mapping and matching at the lexical level.*

2.2.1 Lexical integration

Lexical mapping is an activity that attempts to relate the vocabulary of, at least, two models or two ontologies or... that share the same domain of discourse. Considering ontologies, (Kalfoglou and Schorlemmer 2003) adopt an algebraic approach where an ontology is a pair O = (S, A) where S denotes the ontological signature (it describes the vocabulary) and A denotes a set of ontological axioms (roughly speaking, A expresses the intended interpretation of S). The mapping is then defined as a function that preserves the mathematical structure of ontological signatures and their intended interpretations, as specified by the ontological axioms, i.e. mappings are considered as ontology morphisms (like for instance, a morphism of posets, i.e. a function f that preserves a partial order : a $\leq$ b implies f(a) $\leq$ f(b)). Ontology mappings are then characterized as morphisms of ontological signatures and the mapping may be total or partial where a total ontology mapping from $O_1 = (S_1, A_1)$ to $O_2 = (S_2, A_2)$ is a morphism $f : S_1 \rightarrow S_2$ of ontological signatures. In such a way that, $A_2 \models A_1$, i.e. all interpretations that satisfy O_2's axioms also satisfy O_1's translated axioms.

Moreover, in this domain, there is lot of work from different communities that are relevant to this notion of mapping. Terms and work encountered in the literature include alignment, merging, articulating, fusion, integration, morphism and so on. These words may be classified in three situations, integration, merge and use [85] (Figure 2.6 summarizes these situations) :

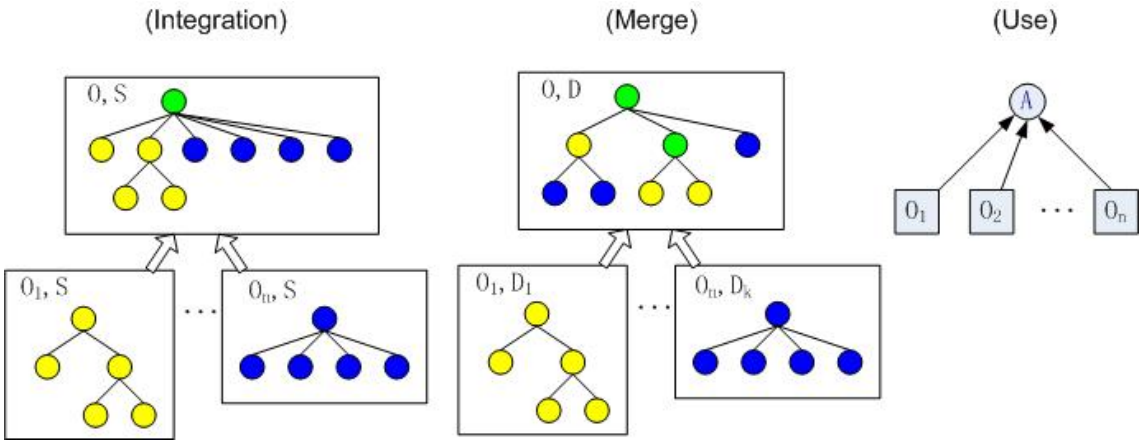


FIG. 2.6 – Three integration models : *integration*, *merge* and *use*

- **Integration** is when we build a new ontology reusing other available ontologies. In integration we have one (or more) ontologies that have to be integrated ($O_1, O_2, \ldots, O_n$ in figure 2.6 : Integration), and on the other hand, the ontology (O) resulting from the integration process. Each ontology, integrated in the resulting ontology, usually is about a different domain other than that of the resulting ontology (D) or the various ontologies integrated $D_1, D_2, \ldots, D_k$).
- **Merge** is when we build an ontology by merging several ontologies into a single one that unifies all of them. In the merge process we have a set of ontologies (at least

two) $O_1, O_2, \ldots, O_n$, that are going to be merged, and the resulting ontology (O) (figure 2.6 : Merge). The goal is to make a more general ontology about a subject by gathering into a coherent bulk, knowledge from several other ontologies in that same domain. The domain of both the merged and the resulting ontologies are the same (D) although some ontologies are more general than others.

– **Use** is when we build an application using one or more ontologies. In use there are one or more ontologies involved ($O_1, O_2, \ldots, O_n$) and there is no resulting ontology. One cannot draw any conclusions as to the architecture of the resulting application because it depends on the application itself.

Another theory of integration has been the focus of study on lexical and figurative meaning [4, 3]. Some online electronic dictionaries have been developed to assist the automatic processing of the natural language ; they support some relationships between the terms (name of *concept*), as name equality, synonyms, homonyms, hyponyms, abbreviations, etc. SUMO (Suggested Upper Merged Ontology) falls into this category : it was promoted by the IEEE Standard Upper Ontology effort [98]. The SUMO was created by merging publicly available ontological content into a single, comprehensive, and cohesive structure. In February 2003, the ontology contains 1000 terms and 4000 assertions. It is implemented in DAML+OIL.

WordNet is the most popular lexical tool to day. It is an extremely large and free online English lexical database [65]. The database is divided in parts of speech into nouns, verbs, adjectives, and adverbs. The nouns are organized as a hierarchy of nodes. In version 2.0 of WordNet, there are 141690 noun synsets, 24632 verb synsets, 31015 adjectives and 5808 adverbs. WordNet is continually updated, and several versions of the database currently used in Information Retrieval and Natural Language Processing applications. EuroWordNet is a multilingual database with WordNets for several European languages (Dutch, Italian, Spanish, German, French, Czech and stonian) [84]. The wordnets are structured in the same way as the WordNet in terms of synsets (sets of synonymous words) with basic semantic relations between them. Each wordnet represents a unique language-internal system of lexicalizations.

2.2.2 Schema Matching

Semantic matching which plays a central role in knowledge integration is mentioned in section 2.2.1. It also has numerous applications, such as Web-oriented data integration, electronic commerce, schema evolution and migration, application evolution, and component-based development. Schema matching is typically performed on semantic matching, which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other where schema can be simply defined as a set of elements connected by some structure[LC94, MIR94, MZ98, PSU98, MWJ99, DDL00] .

Schema matching is a basic problem in many database application domains, such as heterogeneous database integration, E-commerce, data warehousing, and semantic query processing. Most works on schema matching has been motivated by schema integration.

The problem, given a set of independently developed schemas construct a global view, has been investigated [81]. In an artificial intelligence setting, this is the problem of integrating independently developed ontologies into a single ontology. An analysis on seven published prototype implementations is presented in [88], they class the schema matching approaches into three levels.

Schema-level matchers only consider schema information, not instance data. The available information includes the usual properties of schema elements, such as name, description, data type, different kinds of relationships (part-of, is-a, etc.), constraints, and schema structure.

Instance-level data can give important insight into the contents and meaning of schema elements. This is especially true when useful schema information is limited, as it is often the case for semi-structured data.

Combining matchers can be done in two ways, hybrid matchers and the combination of independently executed matchers.

Currently, we can find numerous matching approaches implemented by different methods, and applied in different domains. The SemInt [27] match prototype creates a mapping between individual attributes of two schemas. SemInt uses neural networks to determine match candidates. The LSD (Learning Source Descriptions) system uses machine-learning techniques to match a new data source against a previously determined global schema [33]. It represents a composite match scheme with an automatic combination of match results. It can take self-description input, such as XML, and make its matching decisions by focusing on the schema tags while ignoring the data instance values. The SKART (Semantic Knowledge Articulation Tool) prototype follows a rule-base approach to semi-automatically determine matches between two ontologies [67]. Rules are formulated in first-order logic to express match and mismatch relationships and methods are defined to derive new matches. SKAT is used within the NION [68] architecture for ontology integration. The TransScm prototype [66] uses schema matching to derive an automatic data translation between schema instances. In [78, 29], Palopoli et al. propose algorithms to automatically determine synonym and inclusion (is-a, hypernym) relationships between objects of different entity-relationship schemas. The algorithms are based on a set of user-specified synonym, homonym, and inclusion properties. ARTEMIS is a schema integration tool [93]. It then completes the schema integration by clustering attributes based on those affinities and then construction views based on the clusters. ARTEMIS is used as component of a heterogeneous database mediator. Cupid is a hybrid matcher based on both element and structure level matching [62]. It is intended to be generic across data models and has been applied to XML and relational examples. It uses auxiliary information sources for synonyms, abbreviations, and acronyms. However, the most interesting results were in the value of particular features of each algorithm on particular aspects of the examples, which are too detailed to summarize here.

2.2.3 Semantic Web

The Semantic Web is a web of data, in some ways like a global database. The Semantic Web is about two things. First of all, it is about common formats for data interchange, where on the original Web we only had interchange of HTML (Hypertext Markup Language) documents. Secondly about language to record how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing. In Feb 2004, The World Wide Web Consortium released the Resource Description Framework (RDF) and the OWL Web Ontology Language (OWL) as W3C Recommendations.

RDF is used to represent information and to exchange knowledge in the Web. When looking at a possible formulation of an universal Web of semantic assertions, the principle of minimalist design requires that it is based on a common model of great generality. Any prospective application can be mapped onto the model, only when the common model is general.

OWL is used to publish and share sets of terms which must be annotated in some lexical ontologies, supporting advanced Web search, software agents and knowledge management. Instead of continuing with separate ontology languages for the Semantic Web, a group of researchers, including many of the main participants in both the OIL and DAML efforts, got together in the Joint US/EU ad hoc Agent Markup Language Committee to create a new Web ontology language. This language DAML+OIL built on both OIL and DAML, was submitted to the W3C as a proposed basis for OWL, and was subsequently selected as the starting point for OWL. Current, the OWL language provides three increasingly expressive sub-languages designed for use by specific communities of implementers and users. OWL Lite supports those users primarily needing a classification hierarchy and simple constraint features. OWL DL supports those users who want the maximum expressiveness without losing computational completeness and decidability of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class can't also be an individual or a property, a property can't also be an individual or a class). OWL DL is so named due to its correspondence with Description Logics. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems. OWL Full is meant for users who want maximum expressiveness and syntactic freedom of RDF with no computational guarantees.

Semantic Web languages and standards are used in Web services, which is another important work of W3C. That brings some new sublanguages, as OWL-S. It is a OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form.

So, the lexical integration theories, the semantic matching approaches , and the standards as semantic web sketch a frame of *capability discovery*. The theories, approaches, and standards should be implemented on some actual system architectures. These system architectures will be distributed and heterogeneous. Thus, we will introduce the actual system architectures in the next section.

2.3 Heterogeneous Architecture

In system architecture research field, heterogeneity often appears with distributed ones. Distribution arises from business and human imperatives and has been facilitated by the rise of reliable, high-speed networks, and by standards-based middle ware such as CORBA. Modern systems are also heterogeneous : they comprise sub-systems built on different platforms. Heterogeneity arises for a variety of reasons : physical constraints, application conditions, available technical skills, and history [31]. We defined this work focus on the heterogeneous logical representation environment where we usually have three action situations, as sketches in figure 2.7.

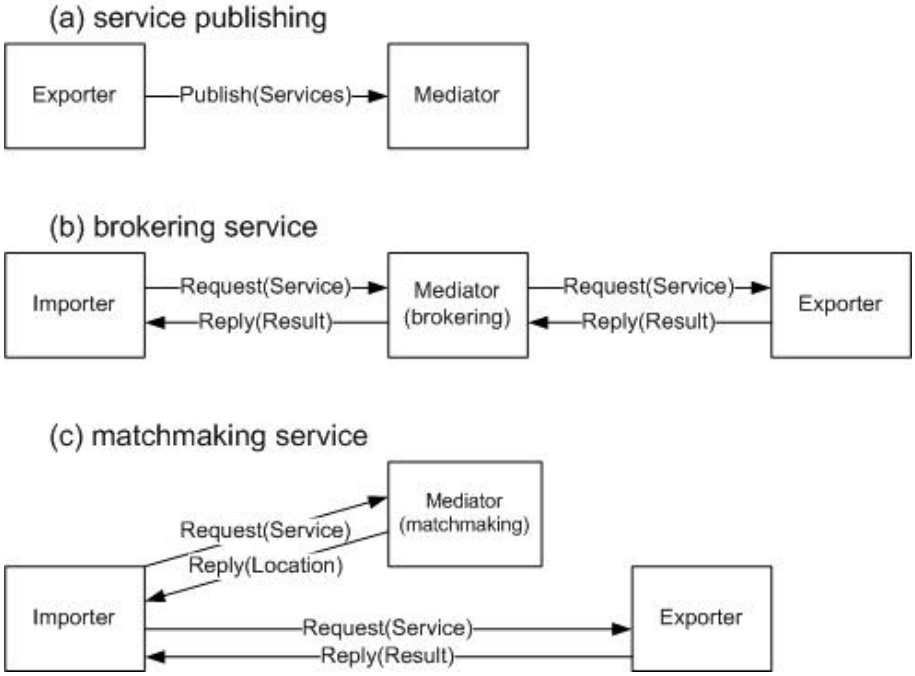


FIG. 2.7 – Three action situations : Service publishing, broking service, and matchmaking service

There are also many mediation roles as presented in figure 2.7. In the three actions, a *mediator* plays a central role. Gio Wiederhold has given a general definition of mediator in 1991 [108]. *A mediator is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications. Exporter* publishes (*tells*) its capabilities in one or more mediators sites (see figure 2.7(a)).

Importer is the knowledge seeker, which sends requests to the mediator asking for exporters fitted with a given set of capabilities. In the brokering action (see figure 2.7(b)), *importer* directly accesses to a *mediator*, where the *mediator* translates the results in the local language between the *importer* and the *exporter*.

In the matchmaking action, the *importer* accesses the *mediator* to get the information of the *exporter* that contains the semantic representations in local language of the user's knowledge in the form of taxonomies, category structures or ontologies. These conceptual actions are independent and they can be implemented in multiple system architectures.

2.3.1 Web Service and Service-Oriented Architecture

Recently, Web services appear to be the most active research in system architecture domain. In February 2002, the W3C Web Services Architecture Working Group exchanged nearly 400 emails over two weeks trying to define the term. The last definition that the team produced was, "*A Web service is a software application or component identified by a URI, whose interfaces and binding are capable of being described by standard formats and supports direct interactions with other software applications or components via Internet-based protocols*".

Web services are designed to support application-to-application communication without human assistance or intervention. Even though Web services are new, from an architectural perspective they are new implementations of service-oriented architecture (SOA). Figure 2.8 sketches the three conceptual roles and operations of a SOA. The three basic roles are the service provider, the service broker, and the service consumer. A service provider makes the service available and publishes the contract that describes its interface. It then registers the service with a service broker. A service consumer queries the service broker and finds a compatible service. The service broker gives the service consumer directions on where to find the service and its service contract. The service consumer uses the contract to bind the client to the service.

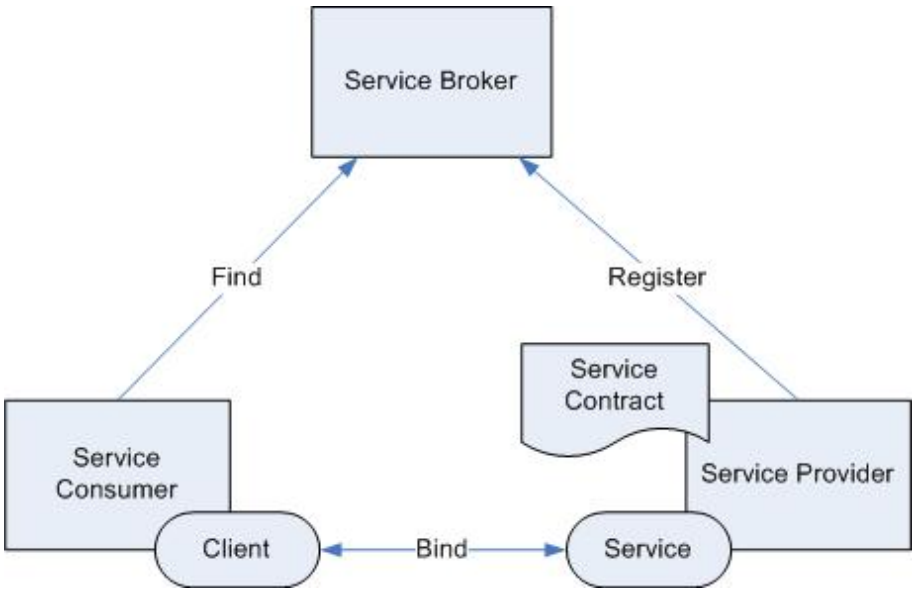


FIG. 2.8 – The conceptual roles and operations of a SOA

Web service application is founded on a group of standard internet protocols, that is different from the middle ware of previous SOA system, such as DCOM, RMI and CORBA. Web services can be developed using any programming language and work on any platform because they all speak the same language, Extensible Markup Language (XML), which is used as the message data format, and is also used as a foundation for all other Web services protocols. Hence three XML-based technologies have emerged as *de facto* standards for Web services :

- Simple Object Access Protocol (SOAP) defines a standard communications protocol for Web service [94].
- Web Services Description Language (WSDL) defines a standard mechanism to describe a Web Service [56].
- Universal Description, Discovery and Integration (UDDI) provides a standard mechanism to register and discover Web services [100].

Before Web service introduced the group of standard Internet protocols, we have had some standards of distribute and heterogeneous architectures, as OMG.

OMG has standardized this process at two key levels : First of all, the client knows the type of the object it's invoking, and the client stub and object skeleton are generated from the same IDL. This means that the client knows exactly which operations it may invoke, what the input parameters are, and where they have to go in the invocation ; when the invocation reaches the target, everything is there and in the right place. We have already seen how OMG IDL accomplishes this. Secondly, the client's ORB and object's ORB must agree on a common protocol - that is, a representation to specify the target object, operation, all parameters (input and output) of every type that they may use, and how all of this is represented over the wire. OMG has defined this also - it's the standard protocol IIOP.

OMG is open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

CORBA is the acronym for Common Object Request Broker Architecture. CORBA defines a service contract in OMG IDL, which is some description on the interface is the syntax part of the contract that the server object offers to the clients that invoke it. Any client who wants to invoke an operation on the object must use this IDL interface to specify the operation it wants to perform, and to marshal the arguments that it sends. The IDL interface definition is independent from programming languages, but maps to all of the popular programming languages via OMG standards : OMG has standardized mappings from IDL to C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript.

The other development technologies can be used in SOA architecture implementation, such as microsoft COM, .NET, and JXTA. And all these technologies can be used in other distributed heterogeneous architectures implement as Mediation and Peer to Peer.

2.3.2 P2P and Distributed Knowledge Management

There are several definitions of Peer to Peer (P2P) architecture [12] [70] : popularly, P2P supports mechanisms that allow end users to participate in distributed information networks without much technical support and heavy-handed computing platforms. Comparing with Client-Server architecture, P2P architectures enables efficient service exchanges by bringing services closer to the point where they are actually consumed, thereby acting as a service caching mechanism.

In the distributed knowledge management approach, a great emphasis is put on autonomy and coordination aspects, so that every knowledge network can manage it's knowledge, exchange knowledge through meaning negotiation, and cooperate with other knowledge networks in order to achieve it's goals. Compared to these aspects a peer to peer system is particularly suitable since depicted with the following capabilities :

- supports autonomy : every member of the system is seen as a peer that manages and has control over a set of local technologies, applications and services ;
- is dynamic : peers and resources can be added or removed at any time ;
- is decentralized : the community of peers is able to achieve its goal independently from any specific member or component ;
- is cooperative : in order to join and use the system, every member must provide resources or services to the others.

All peer-to-peer architectures have one thing in common : the actual data transfer is always peer-to-peer : a direct data connection is made between the peer offering the file and the requestor. The control plan however is implemented in various ways. Figure 2.9 gives an overview of the three types we identified and a typical search-download sequence in which the leftmost peer searches for a file and downloads it from the rightmost peer. Every individual peer-to-peer application uses one of these architectures, with its own specific quirks.

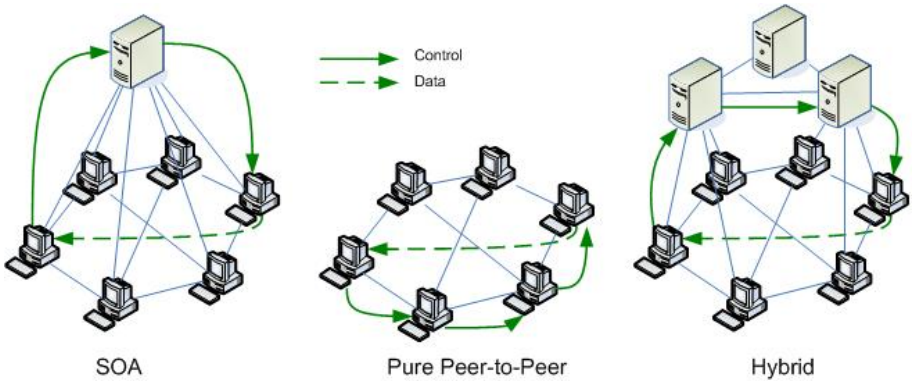


FIG. 2.9 – Three peer-to-peer architectures : *SOA*, *Peer2Peer* and *Mediation*

As we mentioned in the previous section, SOA architecture uses a client-server setup for its control operations. All peers log on to a central server that manages user databases. Searches for a file are sent to the server and, if found, the file can be downloaded directly from a peer. In most cases the server will have a database of files shared by peers. However with the number of court cases against server-based peer-to-peer applications developers are hesitant to use this architecture.

Pure peer-to-peer applications will not use a central server at all (except possibly for logging onto the network). Queries for files can be flooded through the network or more intelligent mechanisms can be used [2]. Pure peer-to-peer networks have become quite

unpopular because they generate a lot of overhead traffic to keep the network up and running.

Mediation architectures is a hybrid architecture. Through the introduction of the so-called mediators, mediation architectures have properties of both the mediated and the pure architectures. The mediator will perform the task of a server in the mediated architecture, but for only a subset of the peers. The mediator themselves are connected through a peer-to-peer network. Mediation architecture was already argued in knowledge management domain, we will move to mediation architecture in the next section.

2.3.3 Mediation architecture

In the last decade, mediation systems have become the reference architecture to integrate both structured and semi-structured data [80, 108, 109]. Such systems are characterized by the presence of mediators that are responsible for a service provider discovery in the entire mediator federation and for the translation, between mediators and service providers, from local languages to a common language (see figure2.10).

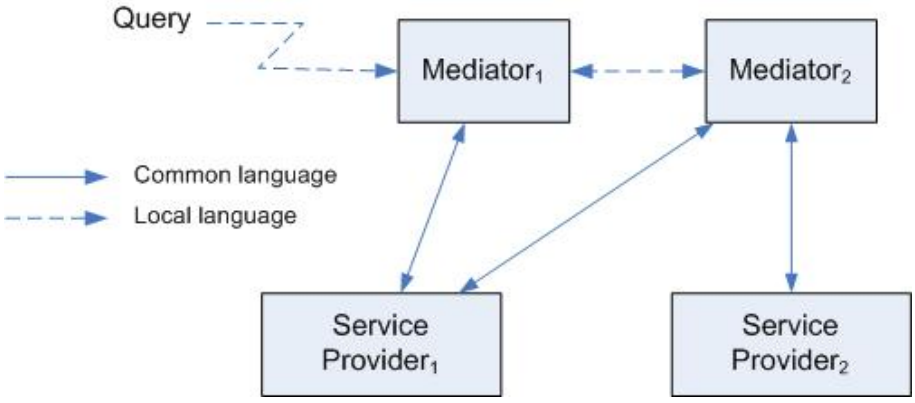


FIG. 2.10 – Mediation Architecture

Mediation architectures are designed to solve two types of problems. The first one is, for single data resources ,a primary hindrance for that the end-user needs to understand the representation of data. The second one is for multiple data resources. The major concern is the mismatch encountered in information representation and structure when combining information from multiple data sources. Mediators have an active role in integrated systems, containing knowledge structures, to drive transformations and to store intermediate results.

In particular, the Intelligent Information Integration approach [40, 108] focuses on solving both the heterogeneity and dynastic of distributed information systems. It creates a central layer by distinguishing the function of mediation from the user-oriented processing and from data resources access. Most user tasks will need multiple, distinct mediators for their subtasks. A mediator uses one or several data resources (see figure2.10). The original query is reformulated in several sub-queries, but this set of sub-queries is hard to be equivalent to the original query [101]. Obviously, the best solution of rewriting query,

which is closest to the original query, is the notion of problems maximally contained in the rewritten query [10, 101]. Indeed, a mediator must have appropriate mechanisms to rewrite queries, which can accurately represent the requirements of an original query, on the other hand, which can accord with the selected service provider.

2.4 Conclusion

Our study is founded on the knowledge representation and management theories, allowing *capability descriptions* in multiple knowledge representation languages to be interchanged between heterogeneous systems. This chapter presented and compared several typical knowledge representation languages, including the DL languages family. DL will be the main knowledge representation in the following chapters, due to its foundations and to the availability of the notion of complement concept that we feel well suited for the identification of gaps between given capabilities and requested ones. *This notion of complement is the foundation of our proposal for the computation of composite answers.* In addition, conceptual graphs and frame logics will be used to illustrate our proposal within a heterogeneous environment, i.e. in the situation we have to cope with capability descriptions expressed in different formalisms either locally or remotely. This facility require techniques to transform a capability description from a formalism into an other one, together with communication between the systems that manage the various capability descriptions. In this framework, this chapter presented and compared popular system architectures and technologies that are candidates for the implementation of capability discovery in a heterogeneous environment. We opted for a federation of mediators we implemented using Web services and Service-Oriented technologies, as detailed in chapter 6.

The coming chapter gives an overview of our solution, which includes a conceptual system architecture and the approach for cooperative capability discovery. Therefore the coming chapter can lighten all the corpora studied in the chapters 5 and 6.

Chapitre 3

Overview of Capability Application in Heterogeneous Environment

Our main goal is to study and to implement an interoperable query service between heterogeneous knowledge systems. Semantic communication in heterogeneous knowledge environment was proved by a lot of research working on ontology. The ideal of this thesis work is to use "capability requirements" as queries in heterogeneous knowledge environments. "Capability" is compared with data that describes specific instances and events, and knowledge that describes abstract classes. To clarify the distinction between data, knowledge and capability in this thesis we give a definition of capability.

Capability describes abstract relations between classes. Each relation typically relates to multiple classes. Experts are needed to gather and formalize capability. Data can be used to disprove capability.

Capability representation usually uses less terms and syntaxes than data representation and knowledge representation. That avoids the need for creating huge syntactical indexes and to transform knowledge and information into an homogeneous format. The "capability" query is easier to accept by more query services which use heterogeneous knowledge representation languages. The purpose of this work requires formalizing and structuring the "capability" that concerns the attributes of "entities" and relationships between "entities" in a given application domain. Thus, we design a capability description language which benefits from the expressiveness of highly powerful logics and well-defined formal semantics that come with them. We opted for Description Logics (DL) [32, 51] as a base of knowledge representation formalism : it has the notable merit that a single mechanism (the classification mechanism) serves at the same time to build and to query extendible domain descriptions.

As a matter of comparison, in [49], "entities" are designed fragments that are described thanks to keywords, the relationships between the fragments are constituted by metrics that measure the similarity between fragments. In [15], "entities" are object-oriented software components and description logics is used, notably, to describe their intended semantics together with possible constraints involving objects methods.

In [18], "entities" are software objects and the capabilities of a software object are specified giving a syntactic part (signatures of the methods or operations the object offers) and a semantic part expressed as logical expressions (a pre-condition and a post-condition

are associated with every object's method). The syntactic conformance of a method to a query is trivial and the semantic conformance uses theorem proving techniques.

One should notice that description logics has been used in many domains, like the database domain [14, 19][4], not only for querying but also for database schema design and integration [25], for reasoning about queries (query containment, query refinement, query optimization and so on) [10]. From our concern, description logics is used mainly for query purposes with the special objective to produce more than "Yes or No" results, based on the complement concept.

3.1 Conceptual Mediation Architecture

From a system architecture point of view, we choose a trader (also called mediator) based architecture [76] very similar to the notion of *discovery agency* in the Web services architecture. In this architecture, an "entity", called *exporter*, publishes its capabilities at one or more mediators sites (see figure 3.1). Entities, called *importers*, send requests to the mediator asking for exporters fitted with a given set of capabilities.

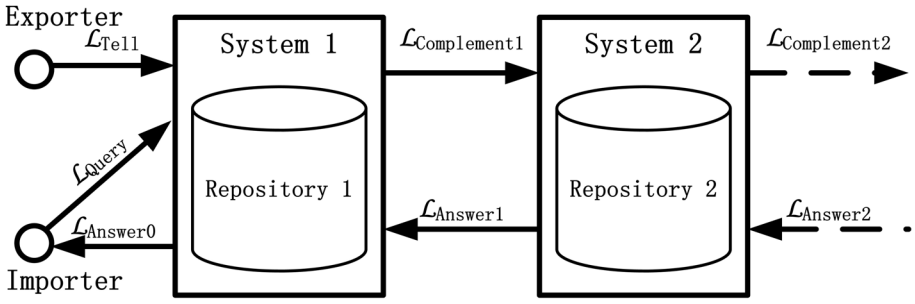


FIG. 3.1 – The Mediator-based Architecture

These capabilities (or services) descriptions are noted in the repositories of the mediators, where are also kept descriptions of *concepts* and *individuals*. To represent the management and the operations on the capabilities and the *concepts* in the repository, we introduce a more formal way of representation, that is used in knowledge base management system (KBMS). Two types of operation of KBMS are *Tell* and *Ask*.

In figure 3.1 and in the following, let $\mathcal{L}$ denote a knowledge representation language. $\mathcal{L}_{Tell}(E)$ denotes that an entity description is written in a given *Tell* language $\mathcal{L}_{Tell}$. Similarly, $\mathcal{L}_{Query}(Q)$ denotes that a query description is written in a *Query* language$\mathcal{L}_{Query}(Q)$, and the query results, expressed in an answer language $\mathcal{L}_{Answer}$, are denoted as $\mathcal{L}_{Answer}(Q)$. So, the *Tell* action and the *Query* action are defined as follows.

Definition 3.1.1. ***Tell action*** *is used to build or to modify the knowledge base (KB) :*

$$TELL : \mathcal{L}_{Tell}(E) \times KB \rightarrow KB'$$

[4]See also [32] for the proceedings of the various "Knowledge Representation meets DataBase" (KRDB) Workshops.

Definition 3.1.2. ***Ask action*** *allows finding the information*

$$ASK : \mathcal{L}_{Query}(Q) \times KB \rightarrow \mathcal{L}_{Answer}(Q)$$

We will detail these operations in the following paragraphs, where we will define the roles in the conceptual mediation model.

3.1.1 Exporter

The exporter is the first role in the conceptual mediation architecture. In this part, we will give a general definition of an exporter in that conceptual model.

Definition 3.1.3. ***Exporter*** *is the provider of a capability. It publishes its capability by* Tell *action at one or several mediator sites.*

This *Tell* action is described in a language $\mathcal{L}_{Tell}$ (see figure 3.1), which can be any description language common between the exporter and the mediator, like the UDDI Registry in Web services.

$\mathcal{L}_{Tell}(E)$ has to contain two types of information. Firstly, it defines an abstract capability description. Multiple exporters can offer the same type of capabilities, and one exporter can offer multiple types of capability. Secondly, it defines the "entity" of the provider, including its name and some basic contact information. We will not talk a lot of "entity" information in this conceptual model. A real application, contains a lot of service interface information, such as service binding template, task management, protocols, etc. These contents will be mentioned in chapter 6.

3.1.2 Importer

The *exporter* is the second role in the conceptual mediation architecture.

Definition 3.1.4. ***Importer*** *is a seeker role for capability, knowledge and data. Importer is a normal software agent role with two description technologies in the conceptual model. It accepts a pair of languages related to an "ask action" :* $\mathcal{L}_{Query}$ *and* $\mathcal{L}_{Answer}$.

It sends requests, using the *Query* operation, to the *mediator* asking for exporters fitted with a given set of capabilities, and then waits for the result by the *Answer* from the mediator. The language $\mathcal{L}_{Query}$ and the language $\mathcal{L}_{Answer}$ are local languages common between the importer and the mediator.

$\mathcal{L}_{Query}$ allows representing kinds of relational queries in object-oriented terms using objects and properties of objects ; it also allows representing object-oriented queries using classes and hierarchy of classes [13]. Instead of returning plain data, $\mathcal{L}_{Query}(Q)$ queries return the query result(s) in the language $\mathcal{L}_{Answer}$, which is a hybrid result set in accordance with $\mathcal{L}_{Query}(Q)$ situations. In this thesis work, we defined three situations for $\mathcal{L}_{Query}(Q)$ and $\mathcal{L}_{Answer}(Q)$:

1. **Individual discovery** which is exactly like searching into a relational database. $\mathcal{L}_{Query}(Q)$ describes a SQL-Like formula, and the mediator returns a result set of object which is some individual descriptions in $\mathcal{L}_{Answer}(Q)$, where we name the object individual in the logical theory.

2. **Model discovery** which navigates through the *concepts* structures, in which the *concept* is a modeled object. In this work we organize these *concepts* into a subsumption hierarchy in DLs. $\mathcal{L}_{Query}(Q)$ contains a *concept* character description ; the result will include the set of *concepts* in accordance with this character description. We can find some sets of individuals under these *concepts*, that we name individualization of concept discovery. Thus, $\mathcal{L}_{Answer}(Q)$ will contain a set of *concepts* and several sets of related individuals.

3. **Capability discovery** which is the top level discovery operation in this work, where a capability is described by modeled relationships between objects (called *roles*, further). It navigates through the capability structures as we did for the concepts in the model discovery. $\mathcal{L}_{Query}(Q)$ contains a capability characteristic description and the result will include the set of *capabilities* in accordance with the described characteristic. We can find some sets of *concepts* which satisfy the capability characteristic description, that we name conceptualization of capability discovery. Following the model discovery, we can also individualize the results of *concepts*, to be able to get some sets of individuals. Thus, $\mathcal{L}_{Answer}(Q)$ may contain a set of capabilities, several sets of related *concepts* and several sets of related individuals.

Our proposals for dealing with this variety of situations are developed in chapter 5.

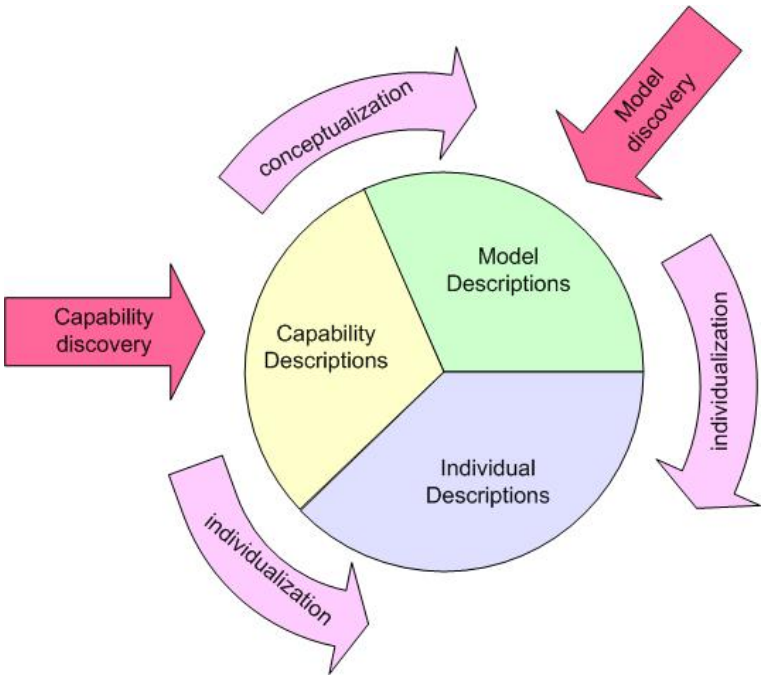


FIG. 3.2 – Capability discovery approach in $\mathcal{ALN}_{r+}$-system

In this conceptual model, it is not necessary that all the importers support all the levels of discovery, but they must describe the requirements in a given capability description language. We will detail the discovery approach in section 5.2.3.

3.1.3 Mediator

Mediator is the core of the conceptual model, as introduced in section 2.3.

Definition 3.1.5. *A **mediator** is [108] a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of application.*

The mediator explores knowledge in its repository to try to satisfy a query issued by an importer. Besides playing the above roles, a mediator has some capabilities such as creating or joining a mediator federation, managing contexts and propagating queries and sub-queries. These capabilities are explained in the following paragraphs.

Mediation Federation

A mediation federation is a group of mediators, which agree to be considered as a sole entity by other mediators when these are requesting services to the former. In other words, an importer can send a query to a mediator in a federation, and the query may be forwarded to each mediator in that federation. As a consequence, the response of the mediator federation is the same as if the query was sent directly to all the members of the mediator federation. The mediator federation can be thought as a "social" aggregation of mediators that display some synergy in terms of content, quality or access strategy. To become a member of a mediation federation, a mediator must provide a mediation federation service, i.e. a capability discovery service.

The mediator implements the capability discovery protocol in $\mathcal{L}_{Query}$ and $\mathcal{L}_{Answer}$, where we will introduce a *complement* concept. The *complement* is a description of unsatisfaction, in a language $\mathcal{L}_{Complement}$, to a pair of $\mathcal{L}_{Query}$ and $\mathcal{L}_{Answer}$. As we sketched in figure 3.1, the *complement* works as a query description between two mediators. So, we get a satisfaction description $\mathcal{L}_{Answer}(Q)$ and a complement description $\mathcal{L}_{Complement}(Q)$ when we send a query Q to a mediator. We see the $\mathcal{L}_{Complement}(Q)$ as part of the answer. We introduce a formula model for this answer in section 3.2, and we will consider the mediation federation's membership strategy in section 6.

Context Management

Context management allows mediator to manage contexts in order to search or to provide information. Contexts provide a semantic classification of the knowledge and data. At the system level, a mediator avoids the need to create huge syntactical indexes and to transform knowledge and information into an homogeneous format ; it gives to each mediator the autonomy to store and maintain its own information in the way it prefers, providing a tool to enable autonomous semantic classification of content.

We opted for capability discovery as a query action in the context management between federated mediators. We know that capability is presented by verbs or verb structures in natural languages, and the number of verbs is much smaller than the number of nouns that are usually used to present the *concept*. Thus, capability discovery supports an easier way of context management. A context manager always trusts more federated mediators which use an homogeneous knowledge representation than those who use heterogeneous knowledge representations.

Query Propagation

The mediation federation is looked at as a dynamic distributed system. A mediator can collaborate with different cooperative mediators for every different query. This functionality allows the mediator system to "use" trust and social relations as a mean to find information in a highly dynamic, heterogeneous and complex environment. In fact a mediator can trust not just the knowledge "content" of another member, but also its "relational" knowledge in terms of the capability, to redirect a request to other trusted mediators.

Therefore, when a mediator receives a query, it might propagate that query to another mediator it considers as an "expert", based on what it believes is the meaning of the request. In order to decide where to propagate a query, a mediator has two possibilities :

1. *a semantic criteria* : if the mediator computes some matching between a query and some *concepts* in its own context, the query resolution mechanism might look for addresses of other mediators that have been associated to the matching *concept*. Here propagation is done on the base of an explicit trust since the provider defines other peers as experts in the query topic ;

2. *a "proximity" criteria* : the query will be sent to known mediators and selection will be done according to some quantitative criteria (number of mediators, number of possible reroutings, hops, etc.) ; this way mediators that are non directly reachable by the importer or that have just joined the system, can advertise their presence and contribute to the satisfaction of the query.

A propagation algorithm may combine the two possibilities and it may be based on a cost function that has the goal to ensure quality and to reduce overflow. This goal is achieved privileging semantic re-routing, and restricting the possibility to perform proximity based hops the more semantic based hops are done. Other parameters and mechanisms controlling the scope of the search and preventing from the message "flooding" are provided : the seeker can set a time to live (TTL), manually limit the number of hops, store in the query the name of mediators that have already received the query, and so on.

3.1.4 Conceptual Mediation Architecture : Concluding remarks

In this conceptual model of mediation architecture, there are three roles : *importer, exporter, and mediator*. We also introduced the capability discovery problem, where the results of a *capability discovery* are described by a composite description : $\mathcal{L}_{Answer}(Q)$ and $\mathcal{L}_{Complement}(Q)$. The next chapters will often refer to this conceptual framework and the chapter 6 contains the description of the choices we made during the development of a system prototype that is intended to validate our proposals. Let us move know to the conceptual model of the query/answer concepts and process which are the foundations of this work .

3.2 A Model for Composite Answer

We intend to design a mediation system to provide a semantic query service in heterogeneous distributed environments. Query/answer process is the core of this work. We propose a composite answer model and we introduce its underlying concept and theory hereafter :

- Section 3.2.1 presents and discusses the notion of query satisfaction,
- Section 3.2.2 formally defines the notion of query complementation,
- and section 3.2.3 shows how given constraints may impact the query evaluation process.

3.2.1 Query Satisfaction

The capability search process is founded on exported capabilities and on relationships between them, these relationships being transparently established by a mediator. When some exporters, known from the mediator, satisfy a query, the references of these exporters are sent back to the importer in $\mathcal{L}_{Answer}$. Nevertheless, query satisfaction may fall into different cases [20] (see figure 3.3) :

- Case1 : There exists exporters where each exporter can exactly satisfy the query ;
- Case2 : There exists exporters where each exporter can fully satisfy the query, and their capabilities are wider than those requested ;
- Case3 : No single exporter that fully satisfy the query exists, but when "combining" or composing capabilities from different exporters, one can fully satisfy the query ;
- Case4 : Neither a single exporter nor multiple exporters satisfy the query, but there exists some exporters that partly satisfy the query ;
- Case5 : No single exporter nor several exporters fully or partly satisfy the query.

One should notice that cases 4 (Partial Satisfaction) and 5 (Failure) would conduct to a failure of a query Q when only one mediator is implied. But, if we assume a grouping of mediators (into a federation of mediators), these cases are typical cases where cooperation among the mediators is required. In the case 5 (Failure), the whole query is transmitted for evaluation to other mediators whereas in the case 4 (Partial Satisfaction), we need to determine "what is missing" to satisfy Q, that means to determine what part of the query is not satisfied by the already found individuals. This "missing part" as well as the original query are transmitted then to a mediator in the federation. Conceptually, we can see the query as being addressed to "the union" of the federated mediators' repositories. Concretely, this union is explored from "near to near" within the mediation federation, that means from a mediator to an other one.

Let us now elaborate more on the conceptual framework and the underlying formal foundations. Given a query Q expressed as a sentence of a query language $\mathcal{L}_{Query}$, a simple result (noted $\mathcal{L}_{Answer}(Q)$) is generally returned in most of the systems and usually $\mathcal{L}_{Answer}(Q) \doteq \{\text{Yes, No, Unknown}\}$, the meaning of "Yes" is that one or several entities

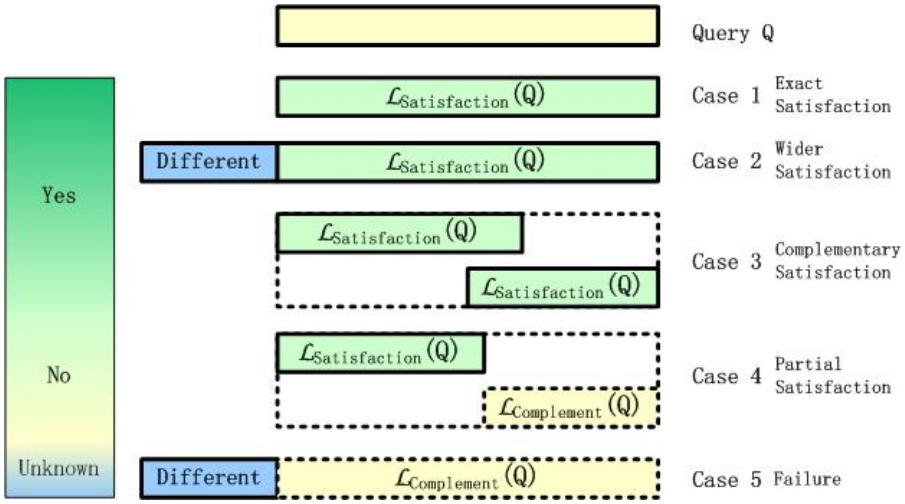


FIG. 3.3 – The five situations of satisfaction

satisfy the $\mathcal{L}_{Query}$ and "No" is the opposite. In this work, we try to offer a $\mathcal{L}_{Answer}(Q)$ that may not be a single entity satisfying $\mathcal{L}_{Query}$, but that may possibly be a collection of entities where "the union" of the members of the collection of entities satisfies the search criteria of $\mathcal{L}_{Query}$. This collection can be found in a repository or in a federation of repositories (that means a set of repositories).

3.2.2 Composite Answer

In this work, we adopted a Description Logic language (DL) [32], that is intensively developed and studied in the field of Knowledge Representation. So it is not surprising that they are particularly adapted for representing the semantics of real world situations including data semantics [25, 14]. The query language is defined by $\mathcal{L}_{Query} \doteq \{\text{DLs formulas}\}$, and $\mathcal{L}_{Answer}$ is defined by two components : $\mathcal{L}_{Satisfaction}$ and $\mathcal{L}_{Complement}$ where $\mathcal{L}_{Satisfaction}$ and $\mathcal{L}_{Complement}$ are two sublanguages of $\mathcal{L}_{Answer}$. Before defining the notion of composite answer, we clarify two definitions : satisfaction and complement.

Definition 3.2.1. *$\mathcal{L}_{Satisfaction}(Q)$ describes a satisfaction, that is a single entity or a set of entities satisfying a query Q. If Q is completely satisfied, its complement (noted $\mathcal{L}_{Complement}(Q)$) is empty. In the contrary, the system will try to determine a complement for this answer.*

For this purpose, we define a function $Comp(-,-)$ to calculate the complement of an answer to a query : $\mathcal{L}_{Complement}(Q) = Comp(\mathcal{L}_{Satisfaction}(Q), Q)$. Intuitively this complement designates "the missing part" to an entity in order for that entity to satisfy the query.

Now that we have mentioned some basic concepts on composite answer, and we sketched the conceptual model of composite answer in figure 3.4, we give the general definition of a composite answer.

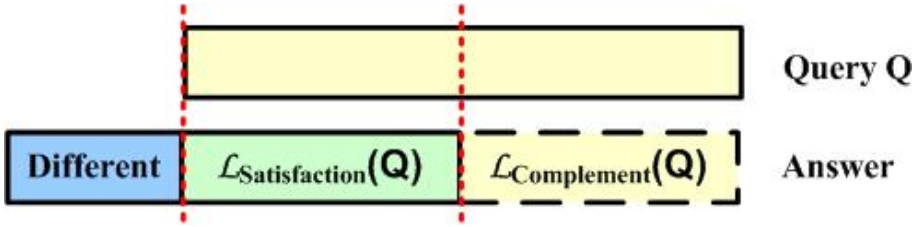


FIG. 3.4 – A model of composite answer

Definition 3.2.2. *When a query is satisfied by a composite answer, the composite answer contains two components : satisfaction and complement, and it allows the satisfaction to contain multiple entity descriptions.*

We can state this definition in a more formal way :

$$\mathcal{L}_{Query}(Q) \subseteq \mathcal{L}_{Answer}(Q);$$
$$\mathcal{L}_{Answer}(Q) \equiv (\mathcal{L}_{Satisfaction}(Q)_1, \cdots \mathcal{L}_{Satisfaction}(Q)_n, \mathcal{L}_{Complement}(Q)).$$

In definition 3.2.2, we said that "satisfied" only means the composite answer logically cover the query. The entities which are described by $\mathcal{L}_{Satisfaction}(Q)$ really satisfy the requirement of the query Q, and the $\mathcal{L}_{Complement}(Q)$) describes the mismatch between $\mathcal{L}_{Satisfaction}(Q)$ and Q. The $\mathcal{L}_{Complement}(Q)$ is difficult to accord with one *concept*, which is a set of attribute/capability of an entity. $\mathcal{L}_{Complement}$ should be a capability representation language.

A composite answer consists of several different answers which hold a part/full satisfaction of the given query. The answers are selected from multiple candidates. The selection makes that the composite answer supports a flexible answer for a given query. There are possibly multiple different composite answers based on one set of candidates for the given query. So, a strategy for the selection of candidate answers may be necessary or imposed by an importer.

3.2.3 Strategies Constraining Composite Answers

In fact, figure 3.3 shows a *"satisfaction first"* selection strategy (which we have applied in our prototype). This strategy always tries to find a full satisfying single/composite answer for a given query. If it cannot find a full satisfying answer, then it will give an answer which holds the "widest" satisfaction for the query. We measure the "wideness" of satisfaction by the capability description number ; we will mention this capability description concept in the chapter 5. In abstract, the required capabilities of the query are satisfied in maximum based on the set of candidate answers.

Following the *satisfaction first* strategy, the situation cases are under the most basic modes of the composite answer, which were sketched by the 5 basic situations : exact satisfaction, wider satisfaction, composite satisfaction, partial satisfaction and no satisfaction at all.

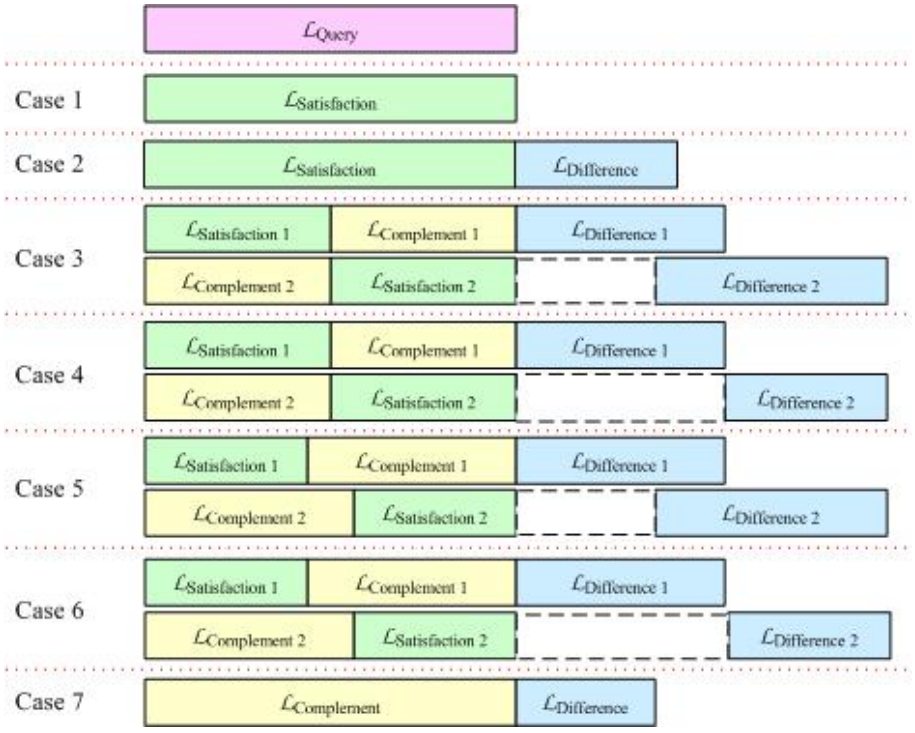


FIG. 3.5 – The nine cases of a composite answer limited to two answers

Additional features may constrain a query and therefore constrain the actual composition of a composite answer. These constraints induce a query evaluation strategy.

For example, one can assert that the answer to a query must not encompass more than two partial and complementary answers, and that the difference between the query and its satisfaction, in case of a wider satisfaction, must be as reduced as possible[5]. As an illustration, figure 3.5 sketches a strategy which imposes that the composite answer must only consist of 1 or 2 entities for a given query. There are 7 cases for a composite answer when we use this strategy. Comparing the two strategies in figure 3.3 and figure 3.5, the cases 1 (exact satisfaction) and 2 (wider satisfaction) are under the same situations, and the case 5 (no satisfaction at all) in figure 3.3 is also the same as the case 7 (no satisfaction at all) in figure 3.5. The cases 3 to 8 in figure 3.5 are different from the cases in figure 3.3. We limit the answers to "at most 2", and we also put the attention on the *Difference* part.

Following the *at most 2 answers* strategy, there are 2 cases of *composite satisfaction* (the case 3 and the case 4), and 2 cases of *partial satisfaction* (cases 5 and 6). In the case 3, the composite answer consists of the two "entities", which when grouped together fully satisfy the query. There are some common parts in the *Difference* between the two entities. From the point view of Knowledge Representation, that means that we use some common vocabularies in the description of the two entities. The common characteristics descriptions relate the two entities in a specific domain. From the point view of an actual

[5]From a concrete application point of view, these types of constraints will enable to express queries like, for example, seeking for at most two partners.

application, the two entities own some common complementary characteristics in a specific domain.

This example emphasizes the need for a strategy to select the answers from a set of candidates. On the other side, the optional strategy makes the flexibility of composite answer. Therefore, we can apply different strategies to satisfy different requirements of the applications, and we can obtain multiple answers for a given query applying different strategies. From a fundamental point of view, dealing with these considerations requires the incorporation into a query language of facilities to express the wished constraints.

However, the situation cases in figure 3.3 will serve as a "standard model" when developing our proposals for the determination of a composite answer in the following chapters. Nevertheless, we will come back to this notion of strategy when we will present the actual algorithm we propose for the calculation of a composite answer (see section 5.2) in order to show how the algorithm's behavior can be influenced by any specified strategy.

3.3 Conclusion

In this chapter, we presented the conceptual mediation architecture, and we detailed some roles and actions in a heterogeneous environment. We also proposed a capability discovery approach in a federation of mediators. We used a notation $\mathcal{L}$ to note the languages in the query/answer actions. There must be some capability description languages in the capability discovery approaches. The ultimate goal of this work is the design and the development of a set of services to export, import and mediate, as well as the study and the development of alternative strategies and mechanisms for mediators cooperation. The coming chapters detail the adopted approach and its foundations.

In the coming chapter, we introduce a capability description language, and the knowledge base in this capability description language, the capability description language being increased from the DL language.

Chapitre 4

Increased Capability Representation Language $\mathcal{ALN}_{r+}$

This chapter focuses on applied Description Logic's language $\mathcal{ALN}$ in knowledge representation, and then it extends the language $\mathcal{ALN}_{r+}$ for capability representation. An initial version of this capability representation language has been proposed in 2006 [34]. We introduced the basic concepts of DLs in section 2.1.1, we will detail these concepts and semantic of expressive languages before any mention on $\mathcal{ALN}_{r+}$.

4.1 Syntax and Semantics of $\mathcal{ALN}_{r+}$

According to the general introduction to the concepts of DLs given in section 2.1.1, DLs-system consist of two components (see figure 2.2) :

1. the knowledge base (KB), which can further be divided into the TBox and the ABox. A TBox stores the conceptual knowledge (vocabulary) of an application domain, while an ABox introduces the assertional knowledge (world description).
2. the reasoning engine, which implements various inference services.

A $\mathcal{ALN}_{r+}$-system comes from a classical DLs-system model. $\mathcal{ALN}_{r+}$ allows an extended syntax for capability description, and then $\mathcal{ALN}_{r+}$-system further divides T_rBox from TBox. A T_rBox specifically stores the capability knowledge (role vocabulary) in the TBox of an application domain.

4.1.1 Concept Description in $\mathcal{ALN}_{r+}$

Before starting with the definition of $\mathcal{ALN}_{r+}$, let us introduce some notational conventions. The letter A, B will often be used for *primitive concepts*, and C, D for *concept* descriptions. Considering *roles*, the letters r, s will often be used for *primitive roles*, the letters R, S for role descriptions, and the letters f, g for role functional restrictions. Non negative integers (in number restrictions) are often denoted by n, m, and individuals are denoted a, b, c, d.

With these notations, *concept* descriptions and capability descriptions are formed according to the syntax rules depicted in figure 4.1. This figure contains a syntax list of

$\mathcal{ALN}_{r+}$-*concept* description, and the lower part is a syntax list of $\mathcal{ALN}_{r+}$-role description.

Name	Abstract syntax		Concrete syntax
primitive concept	$C, D \rightarrow$	$A \mid$	`A`
universal concept		$\top \mid$	`TOP`
bottom concept		$\bot \mid$	`BOTTOM`
primitive negation		$\neg A \mid$	`(not A)`
at-least restriction		$(\geq\ n\ r) \mid$	`(atleast n r)`
at-most restriction		$(\leq\ n\ r) \mid$	`(atmost n r)`
concept conjunction		$C \sqcap D \mid$	`(and C D)`
value restriction on roles		$\forall R.C$	`(all R C)`
role name	$R, S \rightarrow$	$r \mid$	`r`
universal role		$\top_{role} \mid$	`TOProle`
bottom role		$\bot_{role} \mid$	`BOTTOMrole`
role disjunction		$R \cup S \mid$	`(or R S)`
symmetric closure		$R^{-} \mid$	`(R`$^{-}$`)`
transitive closure		$R^{+} \mid$	`(R`$^{+}$`)`
reflexive-transitive closure		$R^{*} \mid$	`(R`*`)`
role functional restriction		RfS	`(RfS)`

FIG. 4.1 – Syntax of $\mathcal{ALN}_{r+}$

As we already mentioned, the *concepts* are of two types, primitive and defined *concepts*. *Primitive concepts* may be considered as atoms that may serve to build new *concepts* (the *defined concepts*). Similarly, *roles* may be primitive roles as well as defined roles. In the example of *concept* descriptions in figure 2.3 (on page 42),

`PERSON` $\dot{\sqsubseteq}\top$ or `PERSON` $\dot{\sqsubseteq}$ `TOP`
`SET` $\dot{\sqsubseteq}\top$ or `SET` $\dot{\sqsubseteq}$ `TOP`

`PERSON` and `SET` are primitive *concepts* : they are introduced using the symbol $\dot{\sqsubseteq}$ and they are linked to a "`TOP`" *concept* ($\top$). Intuitively the TOP *concept* is the "most general one" and it contains all the individuals while the `BOTTOM` *concept* ($\bot$) is the most specific one and it is empty.

Concept constructors	$\mathcal{FL}_0^*$	$\mathcal{FLN}^*$	$\mathcal{ALN}^*$	$\mathcal{ALN}_{r+}$
$\top$	x	x	x	x
$\bot$			x	x
$\neg A$	x	x	x	x
$(\geq\ n\ r)$		x	x	x
$(\leq\ n\ r)$		x	x	x
$C \sqcap D$	x	x	x	x
$\forall R.C$	x	x	x	x
Role constructors				
r	x	x	x	x
$\top_{role}$				x
$\bot_{role}$				x
$R \cup S$	x	x	x	x
R^-				x
R^+				x
R^*	x	x	x	x
RfS				x

FIG. 4.2 – The language $\mathcal{ALN}_{r+}$ and relevant sublanguages

```
TEAM       ≐   SET⊓(∀ member.PERSON) ⊓ (≥ 2 member)
               or
TEAM       ≐    (and SET
                    (all member PERSON)
                    (atleast 2 member))

SMALL-TEAM ≐   TEAM⊓(≤ 2 member)
               or
SMALL-TEAM ≐    (and TEAM
                    (atmost 5 member))
```

`TEAM` and `SMALL-TEAM` are defined *concepts*, which are introduced using the symbol $\doteq$. The *and* constructor enables defining *concepts* as a conjunction of *concepts* : these *concepts* are the immediate ascendants of the defined one. The *all* constructor constrains a role's range and the *at-least* and *at-most* constructors enable specifying the role's cardinality. Finally, the *not* constructor only applies to primitive concept. As mentioned in section 2.1.1, this example is in $\mathcal{ALN}$ (the semantic of $\mathcal{ALN}$ is given in the next section).

Now, let us consider the extension $\mathcal{ALN}_{r+}$: the part of $\mathcal{ALN}_{r+}$-*concept* description (in figure 4.1) is exactly as the $\mathcal{ALN}$-*concept* description syntax. Some extended $\mathcal{ALN}$-role descriptions are mentioned and proved in [6], like $\mathcal{ALN}^*$ which extends $\mathcal{ALN}$ by the role constructors. Figure 4.2 contains a list of all constructors allowed in $\mathcal{ALN}_{r+}$-*concept* descriptions and defines some relevant sub-languages of $\mathcal{ALN}_{r+}$).

4.1.2 Semantic of $\mathcal{ALN}_{r+}$

In order to define a formal semantics of $\mathcal{ALN}$-*concepts*, we consider *interpretation*. An interpretation $\mathcal{I}$ is a tuple $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consists of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns *(i)* to every *concept* name A, a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, *(ii)* to every attribute name a, a partial function from $\Delta^{\mathcal{I}}$ into $\Delta^{\mathcal{I}}$, and *(iii)* to every role name, r, a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Definition 4.1.1. ***(Semantic of $\mathcal{ALN}_{r+}$)*** *The interpretation function is extended to* concept *descriptions in $\mathcal{ALN}_{r+}$ by the following inductive definition :*

$$
\begin{array}{rcl}
\top^{\mathcal{I}} & := & \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} & := & \emptyset, \\
(\neg A)^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}, \\
(\leq n\ r)^{\mathcal{I}} & := & \{a \in \Delta^{\mathcal{I}} \mid |\{b|(a,b) \in R^{\mathcal{I}}\}| \leq n\}, \\
(\geq n\ r)^{\mathcal{I}} & := & \{a \in \Delta^{\mathcal{I}} \mid |\{b|(a,b) \in R^{\mathcal{I}}\}| \geq n\}, \\
(C \cap D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(\forall r.C)^{\mathcal{I}} & := & \{a \in \Delta^{\mathcal{I}} | \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}, \\
\top_{role}^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}, \\
\bot_{role}^{\mathcal{I}} & := & \emptyset \times \emptyset, \\
(R \cup S)^{\mathcal{I}} & := & R^{\mathcal{I}} \cup S^{\mathcal{I}}, \\
(R^{-})^{\mathcal{I}} & := & \{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}}\}, \\
(R^{+})^{\mathcal{I}} & := & \bigcup_{i=1}^{\infty} (R^i)^{\mathcal{I}}, \\
(R^{*})^{\mathcal{I}} & := & \bigcup_{i=0}^{\infty} (R^i)^{\mathcal{I}}, \\
(RfS)^{\mathcal{I}} & := & R^{\mathcal{I}} f S^{\mathcal{I}}
\end{array}
$$

In this definition, R^i denotes the composition $R \circ \ldots \circ R$ of length i, and f is a limited predicate logical description. It is under the model in figure 4.3, and we more precisely define it in the coming section.

4.1.3 Role Description in $\mathcal{ALN}_{r+}$

To give examples of what can be expressed in $\mathcal{ALN}_{r+}$, consider the atomic *concepts* `PERSON` and `FEMALE`. Then `PERSON` $\sqcap$ `FEMALE` and `PERSON` $\sqcap \neg$`FEMALE` are $\mathcal{ALN}_{r+}$-*concept* descriptions, indicating those persons that are female, and those that are not female. If, in addition, we suppose that the *role* `has-child` is an atomic *role*, we can form the *concept* `PERSON` $\sqcap \forall$has-child.`FEMALE` to denote those persons whose all of their children are female. Using the bottom *concept*, we can also describe those persons without a child by the concept `PERSON` $\sqcap \forall$`has-child`.$\bot$.

Further, the added value of transitive closure in individual capability composition is shown in [57]. For example, the $\mathcal{ALN}_{r+}$-*concept* description

$$\texttt{Flight} \doteq \forall\, \texttt{has-flight}^{*}.\texttt{AirPort}$$

intuitively describes all air travels that include Non-stop flights and Transfers. In particular, `has-flight`* has to be interpreted as the reflexive-transitive closure of the role `has-flight`, thus representing the role "transfers".

In addition, since airline companies usually provide round-trip tickets, we can simply describe this capability by `has-flight`$^{-}$. Reflexive-transitive and transitive closures are

kinds of role properties. They can be determined for any *role*.

The *symmetric*, *transitive*, and *reflexive-transitive* describe the character of one single *role*, but they describe a relationship between multiple *roles* on the view of *capability description*. For example, `has-flight`$^-$ presents that the role `has-flight` is symmetric. It means that there are two kinds of flight, while the one is a forward flight, and the second is a back flight. We suppose that there are two roles : `flight-forward` and `flight-back`.

$$\begin{aligned} \texttt{forward-flight} &\dot{\sqsubseteq} \texttt{has-flight} \\ \texttt{back-flight} &\dot{\sqsubseteq} \texttt{has-flight} \end{aligned}$$

The *forward-flight* and *back-flight* are a couple of symmetric roles. As another example, when we introduce a new role `has-train`, we suppose that it exists a kind of composition character between the role `has-train` and the role `has-flight`. Obviously, these descriptions of relationships between roles are very usable in the inference services. But this symmetric character can not be presented in the current DL description languages. So, we propose a new description restriction between the roles, we call *functional restriction* of roles.

These restrictions enforce the interpretations of roles to satisfy certain properties, such as functionality (*role functional restriction*) and transitivity (*transitive closure*). For example, a composition relationship exists (denoted as $\circ$) between the roles, `has-flight` and `has-train`. The role composition $R^{\mathcal{I}} \circ S^{\mathcal{I}}$ (denoted as $R \cap S$ in some language $\mathcal{L}$), is the most usual composition between binary relations. Its semantic can be defined as :

Definition 4.1.2. ***(composite role)***

$$R \circ S := \{(a,c) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists b\ \forall R\ \forall S.(a,b) \in R^{\mathcal{I}} \wedge (b,c) \in S^{\mathcal{I}}\}.$$

Hence, we can use role functional restriction f to present the role composition relationship.

$$R \circ S \Leftrightarrow RfS,\ \textit{where}\ f = \{(a,c) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists b\ \forall R\ \forall S.(a,b) \in R^{\mathcal{I}} \wedge (b,c) \in S^{\mathcal{I}}\}.$$

Equivalence and *subsumption* are the fundamental relationships that may hold among described $\mathcal{L}$-*concepts*. We extend the subsumption relationship between *roles* in this thesis work. We can express simple functional restriction definitions.

Definition 4.1.3. ***(Equivalence and subsumption of roles)***

$$\begin{aligned} R \equiv S &:= \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \leftrightarrow (a,b) \in S^{\mathcal{I}}\}. \\ R \subseteq S &:= \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow (a,b) \in S^{\mathcal{I}}\}. \end{aligned}$$

The role functional restriction model can be used in many relationships between roles. Hence, this formal f is an open model for the relation between descriptions of roles. Many formal relationship descriptions could be accepted by this model, such as composite, equivalence and subsumption. Note, however, that the composite role cannot be expressed in first-order logic. Thus, in terms of expressive power, first-order formulae and some of the second-order formulae in this thesis work are incomparable. We will particularly introduce this f restriction model in the following.

As previously introduced in section 2.1, *capability description* existed in almost all knowledge representation systems, which supports the *concept*/class description. They use some simple syntaxes and less term definitions. One of the points of this thesis work is the introduction of a rich semantic capability description language $\mathcal{ALN}_{r+}$. We did not extend the syntax to enrich the semantic of role description in $\mathcal{ALN}_{r+}$, hence we propose an open capability relationship description model, f role functional restriction, after observation on the description of relationships between roles.

We suppose that f is a limited predicate logical form in $\mathcal{ALN}_{r+}$ (see definition 4.1.1), which describes the relationship between two *roles* and four related *concepts*, as sketched in figure 4.3. Indeed, the figure 4.3 draws a general model of the functional restriction of roles. Here we use the letters C and D, C' and D' for two couples of *concepts*. There is a *role* R between the concepts C and D, and a *role* S between the concepts C' and D'. Therefore, there are six predicate variables (C, C', D, D', R, S) in this functional representation model. f denotes a predicate function on the six predicated variables. In the presentation of the function f, we can use all standard logical connectives that include $\neg$ (logical not), $\cap$ (logical and), $\cup$ (logical or), $\rightarrow$ (logical conditional), $\leftrightarrow$ (logical biconditional).

The quantifiers $\forall$ (universal quantification) and $\exists$ (existential quantification) can also be used before all the variables. There are 4 variable descriptions (C, C', D, and D'), which present 4 *concept* descriptions. If the quantifiers ($\forall$ and $\exists$) only use in this 4 concept descriptions as in Definition 4.1.3 :

$$R \equiv S \Leftrightarrow RfS,\ where\ f = \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \leftrightarrow (a,b) \in S^{\mathcal{I}}\},$$
$$R \subseteq S \Leftrightarrow RfS,\ where\ f = \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow (a,b) \in S^{\mathcal{I}}\}.$$

The functions f are first-order logical forms in the definitions of *equivalence* and *subsumption*.

The R and S variables present 2 *role* descriptions, which describe the relationship between *concepts* ($r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). Therefore, when we use the quantifiers over the variables R and S, the function f will be a second-order logical description form, like in the definition of *composite role* (see Definition 4.1.2),

$$f = \{(a,c) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists b\ \forall R\ \forall S.(a,b) \in R^{\mathcal{I}} \wedge (b,c) \in S^{\mathcal{I}}\}$$

where we used the quantifiers $\forall$ over the variables R and S. In this limited model, these second-order descriptions are accepted. We visually sketch this limited model in figure 4.3.

That functional relation is viewed as a partial function, as defined in the semantic of $\mathcal{ALN}_{r+}$:

$$(RfS)^{\mathcal{I}} := R^{\mathcal{I}} f S^{\mathcal{I}}$$

To elaborate more on this limited *role* relationship description model we go back to the travel example, where it exists two relationships : `has-way` and `has-flight`. If R identifies the relationship `has-way`, then C and D identify the same *concept* `CITY`. S will identify the relationship `has-way`, C' and D' identify the *concept* `CITY-AIRPORT`. There

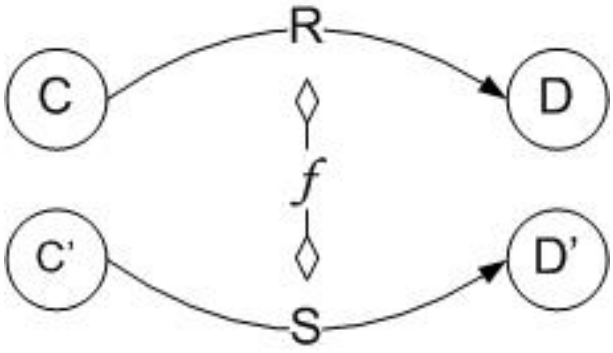


FIG. 4.3 – a model of role functional restriction

is a subsumption in the relationship f between `has-way` and `has-flight`. Subsumption relationship is the fundamental relationship that may hold among described $\mathcal{L}$-*concepts*. Intuitively, a *concept* C subsumes a *concept* D, if the set of individuals represented by C contains the set of individuals represented by D. More formally, C subsumes D and this is denoted $D \sqsubseteq C$ (or D is subsumed by C) if and only if $D^{\mathcal{I}}$ for every possible interpretation $\mathcal{I}$. C is called the subsuming *concept* and D is the subsumed. We can express simple functional restriction definitions (like equivalence and inclusion) by interpretation.

This formal f is an open model for the relation between descriptions of roles. Many formal relationship descriptions could be accepted by this model, such as equalization, subsumption. One could also construct role hierarchies as imposing such restrictions.

Here we will treat role hierarchies in the context of a knowledge base. As we sketch in the figure 4.9 (b), the roles are organized into a role hierarchy by the subsumption relationship in $\mathcal{T}$. Three relationships, `has-way`, `has-flight` and `has-train`, exist in this $\mathcal{T}_r$-Box. The subsumption relationship of roles is a description of functional restrictions f, while the definition of the role subsumption relationship is seen as in the definition 4.1.3.

Of course, using the subsumption hierarchy may also implement all the reasoning services of capability as we did in *concept* hierarchy by the *classification* approaches, as subsumption relationship satisfaction, complement concept determination, etc. As we have said, these roles, or named capability terms, are represented by a verb in a kind of verb or verb-structure in knowledge representation languages. The number of verbs is much more less than the number of nouns in the natural language, while creating a verb common term base is easier than creating a common noun base. On the other hand, the characteristics of subsumption relationship may help the relationship f between roles. For example, if R is composable with S and R' subsumes R, then R' is composable with S. Obviously, it exists a subsumption hierarchy as shown in figure 4.4 (a), we can easily see the role composition relationships in figure 4.4 (b).

So, the subsumption hierarchy can help to find the possible satisfactions for capability discovery. As shown in the example of figure 4.4, the role description `has-way` has two subsumed roles `has-flight` and `has-train`. We can see the lemma of role composition

relationship as :

$$\texttt{has-flight}\circ\texttt{has-train}\rightarrow \texttt{has-flight}\circ\texttt{has-way}$$
and
$$\texttt{has-flight}\circ\texttt{has-train}\rightarrow \texttt{has-way}\circ\texttt{has-train}.$$

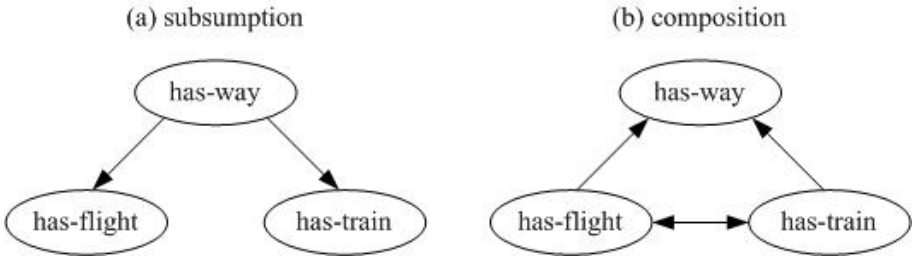


FIG. 4.4 – subsumption and composition

In figure 4.4, we see that it exists some hierarchies of *roles*, after we introduced the limited role functional restriction f. We will implement the *capability discovery* approaches on these hierarchies of roles. We know that the subsumed *concept* description is a full satisfaction for a query in the classification approach, the capability discovery is in the same situation. The subsumed role description will be the full satisfaction. As the example in figure 4.4, from the lemma on subsumption relationship, the subsumer will be a possible satisfaction, i.e., when we cannot find a full satisfaction, we can try to find the satisfaction in the subsumer to compose a composite answer, as we mentioned in the section 3.2.2.

4.1.4 Concluding Remarks

We introduced the syntax and the semantic of $\mathcal{ALN}_{r+}$ in this section. The $\mathcal{ALN}_{r+}$ is an extension of DL languages, where we introduced 3 role closure restrictions : symmetric closure, transitive closure, and reflexive-transitive closure ; and we proposed a limited role functional restriction f. We explained this limited model on relationships between roles. This limited role functional restriction introduces some hierarchies of roles. All these hierarchy structures, and the discovery approaches which have been introduced in the chapter 3 will be implemented in a knowledge representation system, of course which is in language $\mathcal{ALN}_{r+}$. We introduce the knowledge representation system's architecture in the next section.

4.2 KR system based on $\mathcal{ALN}_{r+}$

This KR system is extended from the standard DL knowledge base system, which has been introduced in section 2.1.1. It provides facilities to set up knowledge bases, to reason about their content, and to manipulate them (see figure 2.2). The knowledge base comprises three components, the TBox, T$_r$Box, and ABox. The TBox and T$_r$Box introduce the terminology, i.e., the vocabulary of an application domain, while the ABox contains assertions about named individuals in terms of this vocabulary.

In our work, we distinguished the T_rBox from the standard TBox, to specially introduce the vocabulary of *roles*. These *roles* represent the capabilities of an application domain, while the TBox only stores the vocabulary *concepts* in the application domain (see figure 4.5). The vocabulary *concepts* denote sets of individuals ; and the vocabulary *roles* denote binary relationships between individuals.

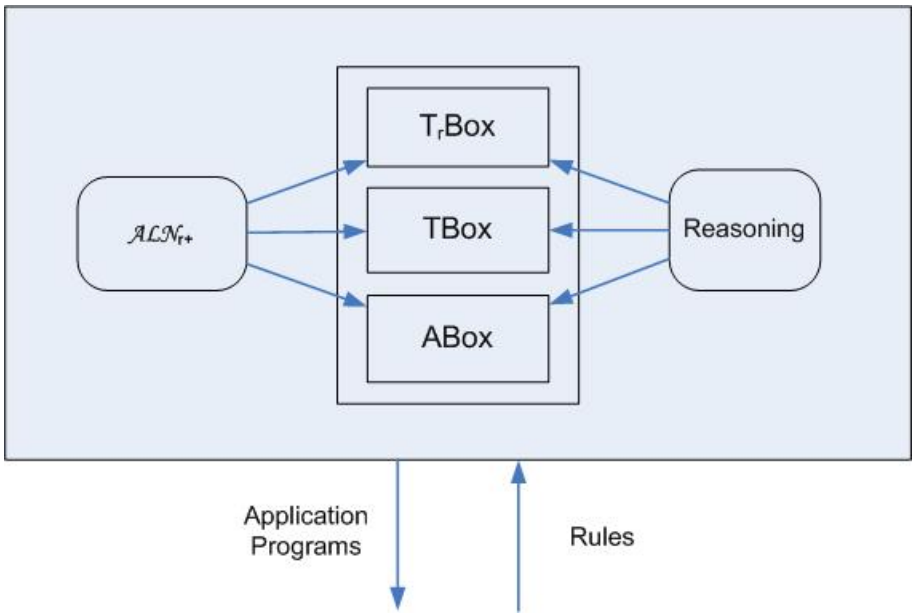


FIG. 4.5 – Architecture of Knowledge Base in $\mathcal{ALN}_{r+}$

This $\mathcal{ALN}_{r+}$-system not only stores terminologies and assertions, but also offers services that *reason* about them. Typical reasoning tasks for a terminology are to determine whether a description is *satisfiable*, or whether one description is more general than another one, that is, whether the first *subsumes* the second. In addition, *concept* description can also be conceived as a query, describing a set of objects one is interested in. Therefore, coupling satisfiability and subsumption with instance tests, one can retrieve the individuals that satisfy the query : this is our conceptual guiding framework (the related algorithms and their implementation are detailed in chapter 5).

From a mediated architecture point of view, as introduced in section 2.3.3, the $\mathcal{ALN}_{r+}$-system is embedded into a large environment. Other federated mediators interact with the KR component by querying the knowledge base and by modifying it, that it, by adding and retracting *concepts*, *roles* and *assertions*. A restrict mechanism for adding assertions uses rules. Rules are an extension of the logical core formalism, which can still be interpreted logically. However, this system provides an application programming interface that consists of functions with well-defined logical semantics, providing an escape hatch by which federated mediators can operate on the KB in arbitrary ways.

This section introduces additional definitions with respect to the TBox (section 4.2.1), the T_rBox (section 4.2.2) and the ABox (section 4.2.3) in an $\mathcal{ALN}_{r+}$ system.

4.2.1 Terminologies and the TBox

The first component of $\mathcal{ALN}_{r+}$ knowledge base are $\mathcal{ALN}_{r+}$-TBox. *Concept* descriptions are used in a TBox to define the *concepts* of the application domain. TBox allows the introduction of names for *concept* description. We restate the definition of an $\mathcal{ALN}_{r+}$-TBox, borrowed from [6, 57].

Definition 4.2.1. ***$\mathcal{ALN}_{r+}$-TBox :*** *An* $\mathcal{ALN}_{r+}$-concept *is of the form* $A \doteq C$, *where* $A \in N_C$ *(N_C being the Concept Name Space) is a* concept *name and* C *is an* $\mathcal{ALN}_{r+}$-concept *description. An* $\mathcal{ALN}_{r+}$*-TBox,* $\mathcal{T}$, *consists of a finite set of* $\mathcal{ALN}_{r+}$-concept *definitions. A* concept *name is called defined (in* $\mathcal{T}$*) if it occurs on the left-hand side of a* concept *definition, otherwise it is called primitive. We require defined names to occur exactly once on the left-hand side of* concept *definitions in* $\mathcal{T}$. *The* concept *description* C *in the definition* $A \doteq C$ *of* A *is called defining* concept *of* A *and is referred to by* $\mathcal{T}(A)$.

We have seen how we can form complex description of *concepts* to describe classes of objects in section 4.1.1, and we also provided terminological axioms in the example on "team" in figure 2.3. In the most general case, terminological axioms have the form,

$$C \dot{\sqsubseteq} D \quad or \quad C \doteq D,$$

where C, D are *concepts*. Axioms of the first kind are called *inclusions*, and they intuitively mean D is more general than C. Axioms of the second kind are called *equalities*. The semantics of axioms is defined by interpretation as follows :

Definition 4.2.2. ***Axioms of*** $\mathcal{T}$ ***:*** *An interpretation* $\mathcal{I}$ *satisfies an inclusion* $C \dot{\sqsubseteq} D$ *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, *and it satisfies an equality* $C \doteq D$ *if* $C^{\mathcal{I}} = D^{\mathcal{I}}$.

If $\mathcal{T}$ is a set of axioms, then $\mathcal{I}$ satisfies $\mathcal{T}$ if and only if $\mathcal{I}$ satisfies each element of $\mathcal{T}$. If $\mathcal{I}$ satisfies an axiom, then we can say that it is a model of this axiom. Some works use the $\equiv$ to specifically denote *equalities* axioms, where we use this symbol to specifically denote the *equivalent* relation between the *concepts*.

Two axioms or two sets of axioms are *equivalent* if they have the same models. Similarly, we say that two *concepts* C, D are equivalent, and write $C \equiv D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretation $\mathcal{I}$.

The preceding definitions can be adapted to roles as introduced hereafter.

4.2.2 Terminologies of Capability and the T$_r$Box

In this thesis work, we propose a new component, $\mathcal{ALN}_{r+}$ T$_r$Box, into $\mathcal{ALN}_{r+}$ knowledge base. By this classical definition, the *role* description is part of $\mathcal{L}$-TBox. The *role* just is an assisted part of *concept* description. In this work, we introduce role space (T$_r$Box) for capability description. Capability descriptions are implemented in a T$_r$Box by the definition of the roles of the application domain. A T$_r$Box allows to introduce names for

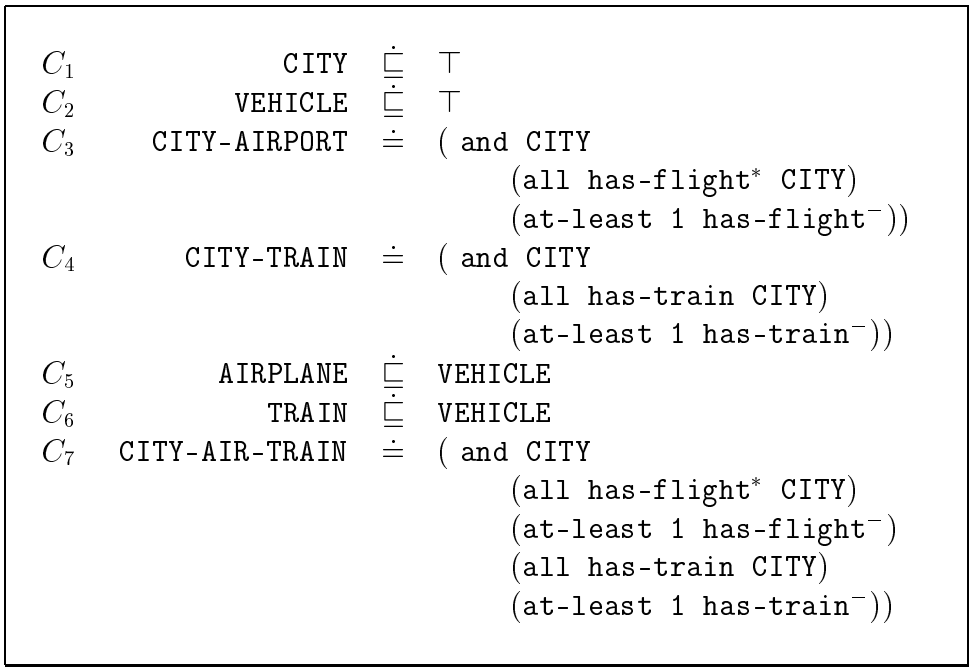


FIG. 4.6 – An example of $\mathcal{ALN}_{r+}$-TBox

role description. Then, an $\mathcal{ALN}_{r+}$-T$_r$Box is defined as follows :

Definition 4.2.3. $\mathcal{ALN}_{r+}$**-T_rBox :** *An $\mathcal{ALN}_{r+}$-role is of the form $r \doteq R$, where $r \in N_r$ (N_r being the Role Name Space) is a role concept name and R is an $\mathcal{ALN}_{r+}$-role description. An $\mathcal{ALN}_{r+}$-T_rBox, $\mathcal{ALN}_{r+} - \mathcal{T}_r$, consists of a finite set of $\mathcal{ALN}$-role definitions. A role name is called defined (in $\mathcal{T}_r$) if it occurs on the left-hand side of a role definition, otherwise it is called primitive. Defined names are required to occur exactly once on the left-hand side of role definitions in $\mathcal{T}_r$. The role description R in the definition $r \doteq R$ of r is called defining role of r and it is referred to by $\mathcal{T}_r(A)$.*

The terminological axiom of $\mathcal{ALN}_{r+} - \mathcal{T}_r$ accords with $\mathcal{ALN}_{r+} - \mathcal{T}$. Indeed, role terminological axioms have a form similar to the one of the concepts terminology :

$$R \dot{\sqsubseteq} S \quad or \quad R \doteq S,$$

where R, S are roles. Axioms of the first kind are called *inclusions*, while axioms of the second kind are called *equalities*. The semantics of these axioms is defined by interpretation as follows :

Definition 4.2.4. ***Axioms of $\mathcal{T}_r$:*** *An interpretation $\mathcal{I}$ satisfies a role inclusion $R \dot{\sqsubseteq} S$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, and it satisfies a role equality $R \doteq S$ if $R^{\mathcal{I}} = S^{\mathcal{I}}$.*

$\mathcal{I}$ satisfies a set of axioms $\mathcal{T}_r$ if and only if $\mathcal{I}$ satisfies each element of $\mathcal{T}_r$. And two axioms or two sets of axioms are *equivalent* if they have the same models.

Let's move to an simple example (see figure 4.7) of the $\mathcal{ALN}_{r+} - \mathcal{T}_r$ of application on travel. We add a definition into the T$_r$Box $\mathcal{ALN}_{r+} - \mathcal{T}_r$:

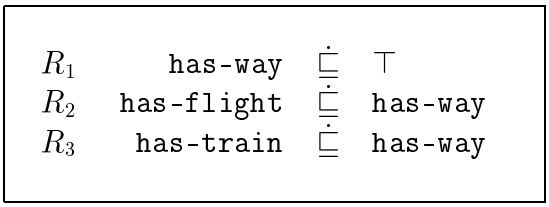


FIG. 4.7 – An example of $\mathcal{ALN}_{r+}$-T$_r$Box

In figure 4.7, intuitively, the T$_r$Box says that `has-way` role has two sub *roles* `has-flight` and `has-train` and they are *atomic roles*.

4.2.3 World Description and the ABox

The third component of $\mathcal{ALN}_{r+}$ knowledge bases is the $\mathcal{ALN}_{r+}$-ABox. Whereas $\mathcal{ALN}_{r+} - \mathcal{T}$ restricts the set of possible worlds, an $\mathcal{ALN}_{r+}$-ABox allows the description of a specific state of the world by introducing individuals together with their properties.

Definition 4.2.5. ***$\mathcal{ALN}_{r+}$-ABox :*** *A* concept *assertion is of the form $C(a)$ where C is a* concept *description and $a \in N_I$ is an individual. A role assertion is of the form $r(a, b)$ where $r \in N_R$ is a role name and $a, b \in N_I$ are individuals. An ABox $\mathcal{A}$ is a finite set of assertions, i.e.* concept *and role assertions. If all* concept *descriptions occurring in $\mathcal{A}$ are $\mathcal{ALN}_{r+}$-*concept *descriptions, then $\mathcal{A}$ is called $\mathcal{ALN}_{r+}$-ABox. In case $\mathcal{A}$ is defined with respect to a TBox, then the* concept *and role descriptions in $\mathcal{A}$ may contain defined names of the TBox and T$_r$Box.*

The semantics of $\mathcal{ALN}_{r+} - \mathcal{A}$ is defined by interpretation as :

Definition 4.2.6. *An interpretation $\mathcal{I}$ is a model of an ABox $\mathcal{A}$, if all* concept *assertions $C(a) \in \mathcal{A}$ and role assertions $r(a, b) \in \mathcal{A}$ in A are satisfied, i.e., $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$.*

Using *concepts* C and *roles* R, one can make assertions of the two following kinds in a ABox : $C(a)$ *and* $R(a, b)$. In the first type, called *concept assertions*, one states that a belongs to (the interpretation of) C. In the second type, called *role assertions*, one states that b is a filler of the role R for a.

Considering the TBox depicted in figure 4.6, the ABox in figure 4.8 states that `Paris` is a city with an `Airport` and that there is an air connection (`has-flight`) from `Paris` to `Beijing`.

The description of individuals is not only discovered by model discovery or capability discovery. They can calculate some capability composition of individuals in accordance with the some capability descriptions. In the syntax definition of $\mathcal{ALN}_r$, we introduced three closure symbols for role description. They are :

R^- (symmetric closure),
R^+ (transitive closure),
R^* (reflexive-transitive closure),

```
CITY-TRAIN(Beijing), CITY-TRAIN(Nancy),
CITY-AIR-TRAIN(Beijing), CITY-AIR-TRAIN(Paris),
has-train(Nancy, Paris), CITY-AIRPORT(Paris),
CITY-AIRPORT(Beijing), CITY-AIRPORT(Wuhan),
has-flight(Beijing, Wuhan), has-flight(Paris, Beijing)
```

FIG. 4.8 – An example of $\mathcal{ALN}_{r+}$-ABox

which support the composite capability at the individual level.

For example there is a capability description `has-flight`* in an the ABox $\mathcal{A}$ in figure 4.8. $\mathcal{A}$ states that there are `has-flight(Paris,Beijing)` and `has-flight(Beijing,Wuhan)`, i.e., there are two flight descriptions in this $\mathcal{A}$, one from `Paris` to `Beijing`, and the other one from `Beijing` to `Wuhan`. According to the capability description `has-flight`* (i.e. the reflexive-transitive closure of `has-flight`), we can obtain `has-flight`*(`Paris`, `Wuhan`) as a new composite capability for an individual in $\mathcal{A}$, with very simple reasoning by composing individuals' capabilities.

But the composition of *concepts*' capability will be of higher value in many applications. The symmetric closure is also an important extension of capability description. As an example, we know that airline companies usually provide round-trip tickets. So we can simply describe this capability by `has-flight`*.

Anyway, this composition of *concepts*' capabilities can be implemented in $\mathcal{A}$: we can find a route from `Paris` to `Wuhan`, where no direct transportation exists.

These examples illustrate the fact that the relationship between *roles* is an important information in some situations for capability discovery and composition. Further, all the approaches for discovery and composition have to be supported by the some basic logical inference services in a knowledge representation system : these inference services are the topics of the next section.

4.3 Inference services $\mathcal{ALN}_{r+}$ systems

The main features of $\mathcal{ALN}_{r+}$-based knowledge representation systems are inference services which allow to implicit knowledge from the knowledge explicitly stored in the knowledge base. We already mentioned some inference on classical DLs-system, typically, these are *satisfiable*, *subsumption*, etc. And as said in previous sections, a *concept* description can also be conceived as a query, describing a set of objects one is interested in. Therefore, checking the concept satisfiability in a $\mathcal{ALN}_{r+} - T$ is one of the most important inference service in this thesis work since where we provide a query service.

On the other hand, as already mentioned in section 2.1.1, the basic inference on *concept* expressions in Description Logic is subsumption, typically denoted as $C \sqsubseteq D$. Checking subsumption relationship is also an inference service that serves as a basis in our work, since all other inferences are based on the results of the subsumption relationship testing.

Moreover, we check the subsumption relationship not only between two *concepts* but also between roles since we consider an $\mathcal{ALN}_{r+} - \mathcal{T}$. Both the definitions are given hereafter

4.3.1 Subsumption of concepts

Let $\mathcal{T}$ be a $\mathcal{ALN}_{r+} - \mathcal{T}$[6], we define the subsumption relationship between two *concepts* as :

Definition 4.3.1. *A* concept C *is subsumed by a concept* D *with respect to* $\mathcal{T}$ *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *for every model* $\mathcal{I}$ *of* $\mathcal{T}$ *: this is denoted as* $C \sqsubseteq_{\mathcal{T}} D$.

Intuitively, a *concept* C subsumes a *concept* D, if the set of individuals represented by C contains the set of individuals represented by D. More formally, C subsumes D and it is denoted as $D \sqsubseteq C$ (or D is subsumed by C) if and only if $D^{\mathcal{I}}$ for every possible interpretation $\mathcal{I}$. C is called the subsuming *concept* and D is the subsumed one. For example, in figure 4.6, `VEHICLE` subsumes `TRAIN`, and `CITY-TRAIN` is subsumed by `CITY`.

As we will see [6], many important inferences can be reduced to subsumption. Traditionally, the basic reasoning mechanism provided by DL systems checked the subsumption of *concepts*. This is sufficient to implement also the other inferences, as it can be seen by the following :

For two concepts C, D *we have :*

1. C *is unsatisfiable* $\Leftrightarrow$ C *is subsumed by* $\bot$ *;*
2. C *and* D *are equivalent* $\Leftrightarrow$ C *is subsumed by* D *and* D *is subsumed by* C *;*
3. C *and* D *are disjoint* $\Leftrightarrow$ $C \sqcap D$ *is subsumed by* $\bot$.

Note : These statements also hold with respect to a $\mathcal{T}$.

The reduction of subsumption can easily be understood if one recalls that, for two sets M, N, we have $M \subseteq N$ iff $M \setminus N = \emptyset$. The reduction of equivalence is correct because C and D are equivalent if, and only if, C is subsumed by D and D is subsumed by C. Finally, the reduction of disjointness is just a rephrasing of the definition.

$\mathcal{ALN}_{r+}$ knowledge base organizes all terms into two hierarchies, *concept hierarchy* and *role hierarchy*. The term hierarchies model the semantic subsumption relationship among *concepts* and *roles*. For the *concepts*, it allows the definition of a concept, called *subsumed concept*, as a specialization of another existing concept, called *subsumer concept*. A subsumed *concept* inherits properties and capabilities from its subsumer *concept*, and in addition it may have specific properties and capabilities. Under the same conditions, we also have the notions of *subsumed role* and the *subsumer role*.

[6]For short, we use $\mathcal{T}$, $\mathcal{T}_r$ and $\mathcal{A}$ for $\mathcal{ALN}_{r+} - \mathcal{T}$, $\mathcal{ALN}_{r+} - \mathcal{T}_r$, and $\mathcal{ALN}_{r+} - \mathcal{A}$, respectively.

4.3.2 Subsumption of roles

For an $\mathcal{ALN}_{r+}$ system, we extend the subsumption relationship to *roles*. A role R *is subsumed* by a role S if in every model of $\mathcal{T}_r$ the set denoted by R is a subset of the set denoted by S. The subsumption relationship between two *roles* is formally defined as follows :

Definition 4.3.2. *Let $\mathcal{T}_r$ be a T_rBox. A role R is subsumed by a* role *S with respect to $\mathcal{T}_r$, if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}_r$. In this case we write $R \sqsubseteq_{\mathcal{T}} S$.*

Intuitively, a role S subsumes a role R, if the set of individuals represented by S contains the set of individuals represented by R. S is called the subsuming role and D is the subsumed one. For example, in figure 4.7, `has-way` subsumes `has-flight`, `has-train` is subsumed by `has-way`.

In addition, it is easy to see that some rules of subsumption relationship exists in definition of $\mathcal{ALN}$:

Definition 4.3.3. ***Subsumption rules between roles :***

$$\begin{array}{lcll} R & \sqsubseteq & R^{-} & \textit{rule 1} \\ R & \sqsubseteq & R^{+} & \textit{rule 2} \\ R^{+} & \sqsubseteq & R^{*} & \textit{rule 3} \\ R & \sqsubseteq & R \cup S & \textit{rule 4} \end{array}$$

where $R \sqsubseteq S$ denotes the subsumption relationship between roles, i.e., $R \sqsubseteq S := \{a \in \Delta^{\mathcal{I}} | \exists b.(a,b) \in R^{\mathcal{I}} \rightarrow (a,b) \in S^{\mathcal{I}}\}$.

We recall the semantic description of *role symmetric* in the definition 4.1.1, and we apply the $(R^{-})^{\mathcal{I}} := \{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}}\}$ in the role subsumption definition, we get the rule 1. The other rules can be proved in some similar ways too.

Applying these rules can be easy to checking the subsumption relationship between the roles. For the example in figure 4.7,

`has-train` $\sqsubseteq$ `has-train`$^{-}$	by rule 1
`has-train` $\sqsubseteq$ `has-train` $\sqcup$ `has-flight`	by rule 4
`has-train`$^{+}$ $\sqsubseteq$ `has-train`* $\sqcup$ `has-flight`	by rule 3 and 4

To simulate the subsumption of *concepts*, the basic reasoning mechanism on roles is provided by checking the subsumption of roles. This is sufficient to implement also the other inferences (*unsatisfiability, equivalence* and *disjointness*), as defined by the following.

For two roles R, S we have :

1. *R is unsatisfiable $\Leftrightarrow$ R is subsumed by $\bot_r$;*
2. *R and S are equivalent $\Leftrightarrow$ R is subsumed by S and S is subsumed by R ;*
3. *R and S are disjoint $\Leftrightarrow$ $R \sqcap S$ is subsumed by $\bot_r$.*

These statements also hold with respect to a $\mathcal{T}_r$.

As an illustration, considering the example in section 4.2, figure 4.9 (a) is a *concept* hierarchy, that is built in $\mathcal{T}$, and (b) is a role hierarchy, that is built in $\mathcal{T}_r$, where the two hierarchies are under a lattice structure, that is an acyclic directed graph. Each node represents a *concept* or a role, the line represents the subsumption relationship. For example, C_1 represents the *concept* `CITY`, and C_3 represents the *concept* `CITY-AIRPORT` ;

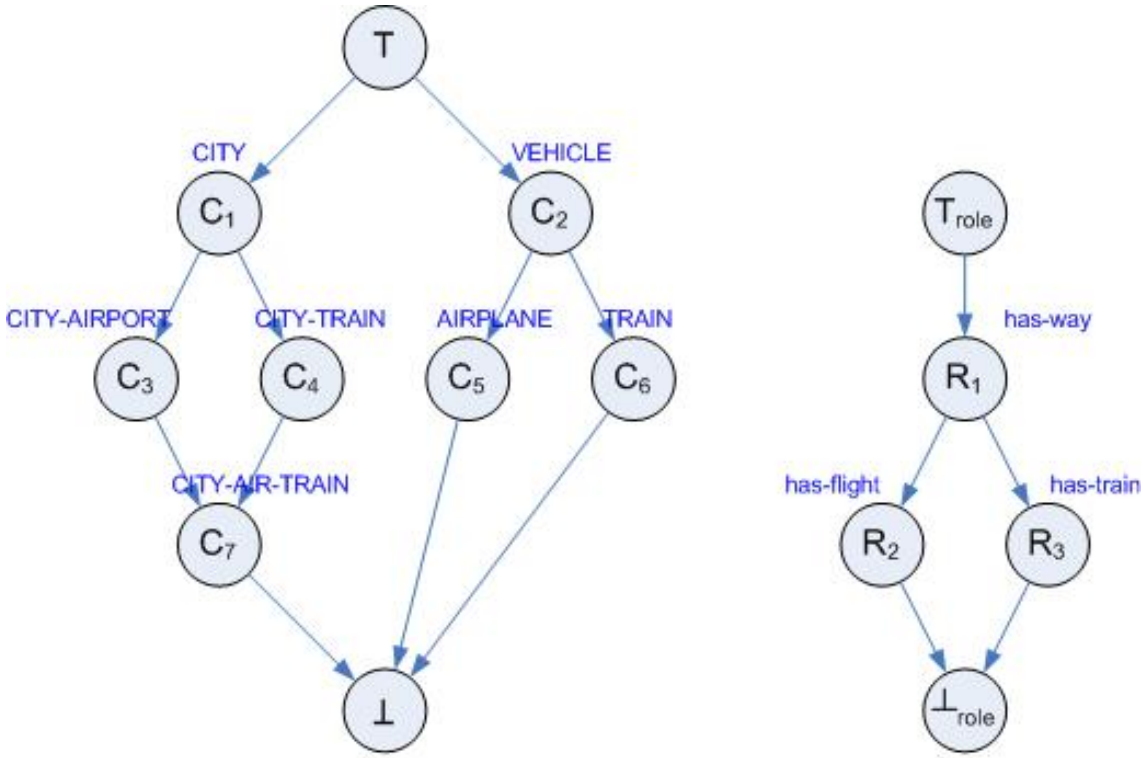


FIG. 4.9 – Terminological Hierarchies

CITY-AIRPORT$\sqsubseteq$ CITY, i.e. the *concept* CITY is the subsumer *concept* of the *concept* CITY-AIRPORT and CITY-AIRPORT is the subsumed *concept*.

4.4 Conclusion

In this chapter, we introduced a capability description language, which is an extension of a "standard" knowledge representation language. We introduced three role closure restrictions (symmetric closure, transitive closure, and reflexive-transitive closure) and we proposed a limited role functional restriction f in the section 4.1. Then we introduced a conceptual model of knowledge representation in $\mathcal{ALN}_{r+}$ in the section 4.2. The last section 4.3 introduced some basic inference services in this knowledge representation system.

In addition, the subsumption relationship organizes the *concepts* and *roles* into a hierarchy of *concepts* and a hierarchy of *roles* (see figure 4.9). An important process, the classification process, aims at determining the position of a new *concept* in a given hierarchy. In this framework, we consider a query as a *concept* Q to be classified in a given hierarchy and the result of the query is the set of instances of the *concepts* that are subsumed by Q. Concepts and roles classification constitutes the foundation for checking whether a subsumption relationship holds or not and therefore it is also the basis for the implementation of the inference services.

These inference services are implemented in this thesis work. So, we move the focus on the algorithms and on the approach for the implementation of these inferences in the coming chapter.

Chapitre 5

Capability Management and Discovery in $\mathcal{ALN}_{r+}$-KB

In chapter 4, we proposed a new knowledge representation language, $\mathcal{ALN}_{r+}$ for capability representation. The $\mathcal{ALN}_{r+}$ increases the skills on *role* description from $\mathcal{AL}$. We also introduced the knowledge base conceptual model in $\mathcal{ALN}_{r+}$ (section 4.2, chapter 4) which includes three components : TBox, T_rBox, and ABox. In chapter 3, we introduced a conceptual view of a capability application in heterogeneous environments, where we defined two principal actions between heterogeneous knowledge bases, *Tell* and *Ask* (see definition 3.1.1 and definition 3.1.2 in section 3.1, chapter 3). In the definition of the *Ask* action, we introduced three *Ask* operations which are *individual discovery*, *model discovery* and *capability discovery*. The implementation part of this thesis concerns the implementation of these operations in heterogeneous knowledge bases.

In this chapter, we introduce the approach and the algorithms that deal with the two steps of a *capability application* : the *capability management* and the *capability discovery*, in an $\mathcal{ALN}_{r+}$ knowledge base.

The *capability management* is developed in section 5.1 and it is based on two hierarchies : a hierarchy of *concepts* and a hierarchy of *roles*. We use the *classification* approach (section 5.1.1) to create and to maintain the hierarchies. The classification process being itself based on the subsumption relationship, we also describe the way we calculate that relationship according to a normalize-and-compare process (section 5.1.2).

The *capability discovery* facilities uses the *capability management* hierarchies to evaluate queries that are addressed to the $\mathcal{ALN}_{r+}$ knowledge base. A noticeable contribution of this work is that we attempt to provide more than a "yes or no" answer to a query. Indeed, when a query, viewed as an $\mathcal{ALN}_{r+}$ concept, cannot be fully satisfied, we identify which part of it is not satisfied and try to make it satisfied either locally or remotely. As already stated, the partial satisfaction is based on the subsumption relationship and on the complement concept. The query satisfaction process is illustrated in section 5.2 while section 5.3 treats the indivudaization process together with its implementation.

5.1 Capability Management for Capability Applications

As sketched in figure 1.3 (page 31), *capability management* includes there sub-fields : *capability representation*, *capability organization*, and *capability reasoning*. We introduced the capability representation language $\mathcal{ALN}_{r+}$ in the previous chapter : it serves for the *capability representation* in this work. We now move the focus on *capability organization* and *capability reasoning*.

Capability reasoning includes some basic inferences, like *satisfaction* or *subsumption*. The algorithms which implement the inference services work on a specific data structure. As many popular knowledge representation systems, we organize the capability descriptions in a terminological hierarchy. In our work, a knowledge base in $\mathcal{ALN}_{r+}$ (figure 4.5, page 78) includes three terminological representation structures : TBox, T$_r$Box, and ABox.

As introduced in the previous chapter, all individual identifications are stored in $\mathcal{A}$, the $\mathcal{ALN}_{r+}$-ABox. $\mathcal{A}$ is a set of individual descriptions, while all individuals are some assertions of *concepts* or *roles*, and one individual identity is in one or several assertions of *concepts* or *roles*. In the *capability discovery* action, we always find the individuals by their modeling, which is possibly a *concept* description or a *role* description. Therefore, in our work, *capability management* strongly relies on terminology management.

Further, we use a *classification approach* [26, 72][7] to organize and maintain terminologies into two hierarchies : a *concept hierarchy* and a *role hierarchy*. The classification process aims at determining the position of a new *concept* in a given hierarchy. In this framework, a query is represented as a *concept* Q to be classified in a given hierarchy. The result of the query Q, when it exits, is the set of instances of the *concepts* that are subsumed by Q. Therefore, classification is a process that enables discovering whether a subsumption relationship holds between a *concept* X and those concepts that are present in a hierarchy $\mathcal{H}$.

In these hierarchies, three concepts play important roles : the *least common subsumer* (LSC), the *most specific concept* (MSC) and the *matching of concept descriptions* (they are introduced and implemented by some non-standard inferences in [57]). Similar concepts are also defined in a hierarchy of roles.

Furthermore, in our work, we opted for a *Normalization-Comparison* algorithm to implement the inferences. The benefit is that it is suited for the management of capability descriptions and *concept* description in $\mathcal{ALN}_{r+} - KB$, as well as for the discovery of capabilities in centralized as well as distributed heterogeneous environments.

This section details our proposals : section 5.1.1 successively elaborates on the classification process in $\mathcal{ALN}_{r+}$-TBox (i.e. concept classification) and in $\mathcal{ALN}_{r+}$-T$_r$Box (i.e. role classification), while section 5.1.2 defines the normalize-and-compare process to determine whether a subsumption relationship holds or not. The material which is presented in this section is used further in section 5.2 to show how we determine a composite answer for a given query.

[7] Originally, the description logics' system classification approaches have been introduced and implemented at the AT&T Labs motivated by applications of the DL-system CLASSIC [7] [8].

5.1.1 Classification

The DLs system classification consists of two important calculation steps : *most specific concept* (MSC) and *least common subsumer* (LCS) [7, 8]. The subsumption test algorithm is the kernel when we calculate MSC and LCS. A. Borgida and P. F. Patel-Schneider have analyzed the correctness of the subsumption algorithm used in description logic-based knowledge representation systems [17]. It provides a variant semantics for descriptions with respect to which the current implementation is complete. We can implement a complement classification approach in a $\mathcal{ALN}_{r+}$-TBox, $\mathcal{T}$. In this section, we will introduce our classification approach in $\mathcal{T}$, then we will expand the classification approach to $\mathcal{ALN}_{r+}$-T$_r$*Box*, $\mathcal{T}_r$.

Classification in $\mathcal{ALN}_{r+}$-TBox

Before detailing the classification approach, we need to define the *most specific concept* (MSC) and the *least common subsumer* (LCS) concepts.

(a) Least Common Subsumer : Intuitively, the LCS of a given sequence of concept descriptions is a description that represents the properties that all the elements of the sequence have in common.

Definition 5.1.1. ***Least Common Subsumer [6] :*** *A* concept *description E in $\mathcal{ALN}_{r+}$ is the* least common subsumer *(LCS) of the* concept *descriptions $C_1, \ldots, C_n$ in $\mathcal{ALN}_{r+}$ (LCS($C_1, \ldots, C_n$), for short) iff it satisfies :*

(i) *$C_i \sqsubseteq E$ for all i, $i \in [1, \cdots, n]$, and*

(ii) *E is the least $\mathcal{ALN}_{r+}$-concept description satisfying C_i, i.e., if E' is an $\mathcal{L}$-concept description satisfying $C_i \sqsubseteq E'$ for all i, $i \in [1, \ldots, n]$, then $E \sqsubseteq E'$.*

For example, in figure 2.3 (on page 42), the *least common subsumer* of the *concepts* `MAN` and `WOMAN` is `PERSON`. In the previous definition, each C_i can be seen as an *atomic concept description*, when the classification approach is applied in the *Ask* action. LCS($C_i, \ldots, C_n$) should be the possible *satisfaction* of the query Q, which is described in $C_i, \ldots, C_n$. In the *Tell* action, the LCS(X) will be the direct superordinate of the new concept X in the concept hierarchy of $\mathcal{T}$.

It is not necessary for the LCS to be a definition of a given concept. In $\mathcal{ALN}_{r+} - \mathcal{T}$, we defined that the concept $\top$ subsumes all the *concepts*. Thus, for a given concept D, $D \sqsubset \top$ always holds, i.e., if we cannot find a LCS for a given concept D, the $\top$ will be the LCS of D. We will be back later to the actual calculation of the LCS in $\mathcal{ALN}_{r+}$.

(b) Most Specific Concept : Originally, the *most specific concept* (MSC) of individuals described in an ABox is a concept description that represents all the properties of the individuals including the concept assertions the individuals occur in, and their relationships to other individuals. If we can find a concept description C in $\mathcal{ALN}_{r+} - \mathcal{T}$, which exactly describes the most specific concept of individuals for a given concept D, we

can say that the concepts C and D are equivalent in this $\mathcal{ALN}_{r+} - \mathcal{T}$ under the closed world assumption. On the other hand, in our approaches of *capability management* and *capability discovery*, the concept descriptions will be normalized, where the normal form of a concept description is a conjunction structure of *atomic* concept which is written as $(and\ C_1, \ldots, C_n)$. An atomic concept, C_i $(for\ all\ i, i \in [1, \ldots, n])$, may not be a *defined concept* or a *primitive concept* : as an example, the atomic concept C_i defined as $C_i \doteq \neg A$, where the concept A is a primitive concept, while the atomic concept C_i is not a primitive concept or a defined concept. A primitive concept or a defined concept has a corresponding set of individuals in the ABox by the definition of DL knowledge base, but an atomic concept does not always find a corresponding set of individuals. So, we propose a definition of Most Specific Concept at the terminological level.

Definition 5.1.2. ***Most Specific Concept :*** *A concept description C in $\mathcal{ALN}_{r+}$ is a most specific concept (MSC) of the concept description D in $\mathcal{ALN}_{r+}$ (MSC(D) for short) iff it satisfies*

(i) $C \sqsubseteq_{\mathcal{T}} D$ *; and*

(ii) *C is the most specific $\mathcal{ALN}_{r+}$-concept description satisfying D, i.e., for all $\mathcal{ALN}_{r+}$-concept descriptions C', if $C' \sqsubseteq_{\mathcal{T}} D$, then it exists $C' \sqsubseteq_{\mathcal{T}} C$.*

Considering the example in figure 5.2, the `TRAIN` concept is one of MSC of the `VEHICLE` concept.

Similarly to the LCS, it is not necessary for the MSC to be a definition of a given concept. Indeed, since the concept $\bot$ is subsumed by all the *concepts* in $\mathcal{ALN}_{r+} - \mathcal{T}$, then, for a given concept D, $\bot \sqsubseteq D$ always holds, i.e., if we cannot find any MSC for a given concept D, then $\bot$ will be the MSC of the concept D.

Intuitively, in the mediator-based architecture, MSC(Q) is a full *satisfaction* of a query Q in the *Ask* action. In the *Tell* action, X being a new concept, the MSCs(X) will be the direct subordinate of the new concept X in the concept hierarchy of $\mathcal{T}$.

(c) MCS, LCS and their use in the classification process The classification approach is also used for *capability management* in [20]. It possibly exists multiple MSC concepts and multiple LCS concepts for a given concept in the classification approach. In the example in figure 5.1 it exists five concept descriptions about programming skills, from *F1* to *F5*. The five concepts describe five kinds of programmer : *F1* knows Java ; *F2* knows Cobol ; *F3* knows Java and Cobol ; *F4* know Java, Cobol, and SQL ; *F5* knows Java, Cobol, and Pascal. We introduces a *role fillers* syntax of concept constructor in DLs for the concept description, which is not included in $\mathcal{ALN}_{r+}$. The *role fillers* syntax is :

`(fillers R` $\mathtt{I}_1 \ldots \mathtt{I}_n$`)` $\qquad\qquad \exists R.I_1 \sqcap \cdots \sqcap \exists R.I_n$

Applying the concept descriptions, *F1* to *F5* can be described as :

```
F1  ≐  (and PERSON (fillers know Java))
F2  ≐  (and PERSON (fillers know Cobol))
F3  ≐  (and PERSON (fillers know Java Cobol))
F4  ≐  (and PERSON (fillers know Java Cobol SQL))
F5  ≐  (and PERSON (fillers know Java Cobol Pascal))
```

In the concept definitions, the `PERSON` is a primitive concept, `know` is a role, and the `Java, Cobol, SQL, Pascal` are four individuals.

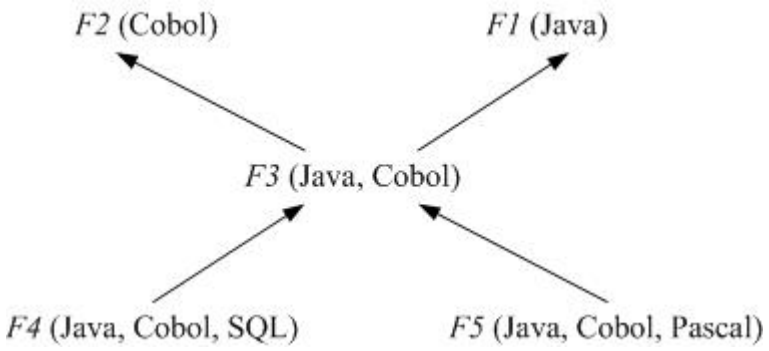


FIG. 5.1 – A (Partial) Example of Software Engineering Hierarchy in [20]

We observe the MSC relationship between two concept descriptions in the definition 5.1.2. We respectively observe the MSC relationships in two groups of concepts : (*F3, F4*) and (*F3, F5*). We see that the concepts *F4* and *F5* are both the most specific concepts (MSCs) of the concept *F3* (MSCs(*F3*) for short). The MSCs(X) are the all direct subordinates for a given concept X in the concept hierarchy of $\mathcal{T}$.

The LCS calculates one concept description for several given concepts in the definition 5.1.1. We observe the LCS relationships for a given single concept in our classification approach, while the relationships actually are the least subsumer Concepts (LSCs) for the given concept.

It possibly exists multiple least subsumers concept descriptions for a given concept. As in the example in figure 5.1, the concept descriptions *F1* and *F2* have the least subsumer relationships with the concept description *F3* (LSCs(*F3*)). Therefore, the LSCs(X) are the all direct superordinates for the given concept X in the concept hierarchy of $\mathcal{T}$.

The calculation of LSCs and MSCs are the foundations of the classification process which is decomposed into 3 steps :

1. Retrieve the most specific concepts of X (denoted $MSCs(X)$) ;
2. Retrieve the least subsumers concepts of X (denoted $LSCs(X)$) ;
3. (possibly) Remove the direct links between MSCs and LSCs of X, and then update the links between X and its MSCs and LSCs.

Figure 5.2 illustrates this classification process. The classification example is based on the concept hierarchy of the TBox $\mathcal{T}$ in figure 4.9 (a). In our mediator architecture,

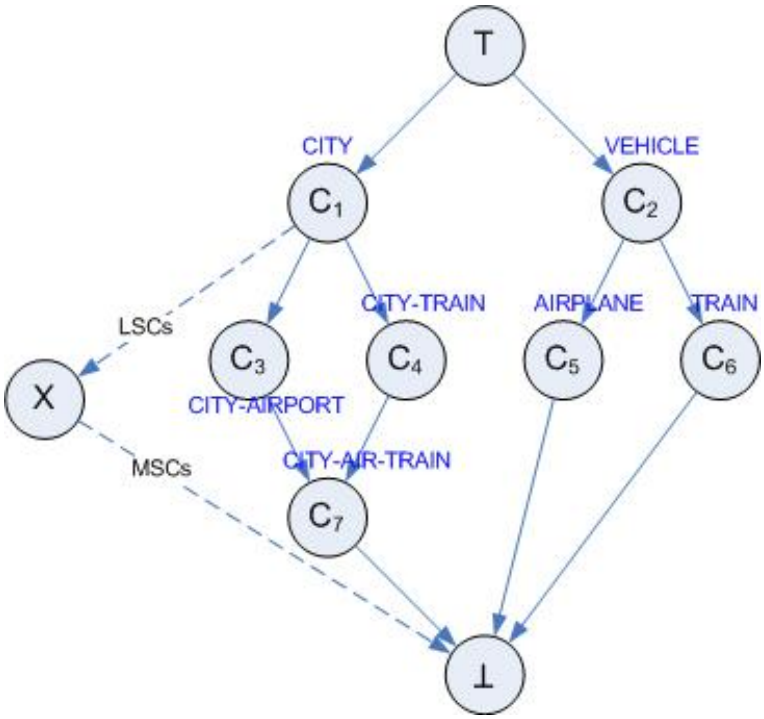


FIG. 5.2 – Adding a concept X into the hierarchy of concept

the classification approach serves the *Tell* action (which we mentioned in section 3.1 on page 60). In the $\mathcal{ALN}_{r+}$ knowledge base system, the *classification* approach is as follows, considering, as an example, the *Tell* action $\mathcal{L}_{Tell}(X)$ where X is defined as

$$X \doteq (\texttt{and CITY}(\texttt{all has-ferry CITY})(\texttt{at-least 1 has-ferry}^{-})).$$

1. Receive the *Tell* action $Tell(\mathcal{L}_{Tell}(X), \mathcal{T})$;
2. The most specific concepts of X does not exist, according to the definition. Therefore, MSCs(X)=$\perp$;
3. Find the least subsumer concepts of X : LSCs(X)=`CITY` ;
4. The direct link between $\perp$ and `CITY` does not exist. Then link the concept `CITY` to X and link the concept description X to $\perp$;
5. Get the new $\mathcal{T}'$.

In the *classification* approach, the result of LSCs(X) includes multiple concept descriptions found in the concept hierarchy of $\mathcal{T}$, when the concept description X is subsumed by a conjunction of concepts, i.e., $X \sqsubseteq C_1 \sqcap \cdots \sqcap C_n$, where the $C_1, \ldots, C_n$ are the LCS *concepts* of the concept description X (In the example of figure 5.2, the *concepts* `CITY-AIRPORT` and `CITY-TRAIN` are the LSCs of the concept `CITY-AIR-TRAIN`).

The result of this action $\mathcal{T}'$ is showed in figure 5.2. We see that the classification approach is an auto-learning mechanism of the $\mathcal{ALN}_{r+}$ knowledge base. We formally note the action as : $Tell(\mathcal{L}_{Tell}(X), \mathcal{T}) = \mathcal{T}'$.

In a query/answer process, if the query is described as the concept description X, the MSCs(X) will be the "best" satisfaction for the query.

However, if MSC(X) only include the $\bot$ concept, the LSCs(X) will be the possible satisfactions. Therefore, MSCs and LSCs are useful concepts for the coverage of the query satisfaction cases we already mentioned (see figure 5.1).

We will elaborate more on the application of LSCs and MSCs in sections 5.2.2 (page 102). We expand the knowledge management methods to *capability management*, and we focus on the classification approach for *capability management* in the coming section.

Classification in $\mathcal{ALN}_{r+}$-T$_r$Box

In a general view, we recur the classification approach on the hierarchy of roles in the T$_r$Box $\mathcal{T}_r$, as we did on the hierarchy of concepts.

The *role* descriptions are properties of *concepts*. The *role* descriptions are noted in the TBox $\mathcal{T}$, but the *role* descriptions are not independent nodes in the hierarchy of concepts in the standard DLs system. We propose a hierarchy of roles in this work, which is a subsumption relationship hierarchy, quite similar to the hierarchy of concepts. Therefore, we can apply the classification approach on the hierarchy of roles. But before applying classification on the hierarchy of roles, we also need to define two concepts : the Least Subsumer Role (LSR) and the Most Specific Role (MSR).

(a) Least Subsumer Role : The *Least Subsumer Role* is defined as follows :

Definition 5.1.3. ***Least Subsumer Role :*** *A role description R in $\mathcal{ALN}_{r+}$ is the least subsumer role (LSR) of the role descriptions S in $\mathcal{ALN}_{r+}$ (LSR(S) for short) iff it satisfies :*

(i) $S \sqsubseteq_{\mathcal{T}r} R$ *;, and*

(ii) *R is the least subsumer $\mathcal{ALN}_{r+}$ role description satisfying S, i.e., for all $\mathcal{ALN}_{r+}$ role descriptions R', if $S \sqsubseteq_{\mathcal{T}r} R'$, then it exists $R \sqsubseteq_{\mathcal{T}r} R'$.*

Regarding the example in figure 5.3, the role description `has-way` is the LSR of the role descriptions `has-flight` and the LSR of `has-train` is `has-way`.

The role description uses a simple syntax in $\mathcal{ALN}_{r+}$ and we defined four rules for testing the subsumption relationship between roles (see definition 4.3.3, page 84). According to the four rules, it is easy to check whether the role subsumption relationship holds or not.

In addition, as mentioned in the previous section, it possibly exists multiple LSCs, or the LSC does not exist for a given concept description. A similar way, multiple Least Subsumer Role descriptions (LSRs(R)) possibly exist for a given role description R. All LSRs(R) are the direct superordinate role descriptions of the role description R in the role hierarchy of $\mathcal{T}_r$.

When the LSR does not exist, we use a special role $\top_{role}$ which subsumes all the role descriptions. Thus, for any role description R, it $R \sqsubset \top_{role}$ always holds, i.e., if we can not find a LSR description for a given role R, the $\top_{role}$ will be the LSR of that role description R.

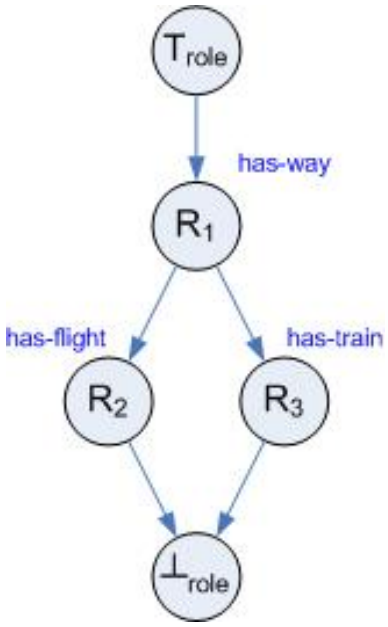


FIG. 5.3 – Example of a Role Hierarchy

As for concept classification, after finding all the superordinates for a given role description, we need to find the subordinates for the given role description. The subordinate role descriptions are named the most specific roles for the given role description in the T_rBox $\mathcal{T}_r$.

(b) Most Specific Role :

Definition 5.1.4. ***Most Specific Role :*** *A role description R in $\mathcal{ALN}_{r+}$ is the most specific role (MSR) of the role description S in $\mathcal{ALN}_{r+}$ (LSR(S) for short) iff it satisfies :*

(i) $R \sqsubseteq_{\mathcal{T}r} S$ *;, and*

(ii) *R is the most specific subsumed $\mathcal{ALN}_{r+}$ role description satisfying S, i.e., for all $\mathcal{ALN}_{r+}$ role descriptions R', if $R' \sqsubseteq_{\mathcal{T}r} S$, then it exists $R' \sqsubseteq_{\mathcal{T}r} R$.*

Alike concepts, we define a special role $\perp_{role}$ which is subsumed by all the *roles* in $\mathcal{ALN}_{r+} - \mathcal{T}_r$. Thus, for a given role R, $\perp_{role} \sqsubseteq R$ always holds, i.e., if we cannot find any MSR for a given concept R, the $\perp_{role}$ will be the MSR of the concept R. Moreover, multiple most specific roles may exist for a given role description. For example, in figure 5.3, the roles `has-train` and `has-flight` are the most specific roles (MSRs) of the role `has-way`.

Intuitively, the MSR(Q) is a full *satisfaction* of a query Q in the *Ask* action. In the *Tell* action, X is a new role to be inserted in the role hierarchy of $\mathcal{T}_r$ and the MSRs(X) are the direct subordinate roles of X in the hierarchy.

(c) Role Classification : The classification of roles is alike the classification of concepts. The calculations of MSRs and LSRs are the two important steps in the role classification process which proceeds as follows :

1. Retrieve the most specific roles of X (denoted $MSRs(X)$) in the role hierarchy ;
2. Retrieve the least subsumers roles of X (denoted $LSRs(X)$) ;
3. (possibly) Remove the links between MSRs and LSRs of X and update the links between X and its MSRs and its LSRs.

Concepts and Roles Classification : Concluding Remarks

The classification approach on concepts serves the *Tell* action, i.e. it helps in the introduction of new concepts in the concept hierarchy. The difference between the concept classification and the role classification is that the classification on roles does not directly serve the *Tell* action in this work, but it is useful for the Ask action, i.e. for capability retrieval. Because one capability is unvalued in this work, when the capability is owned by any entity or individual. The $\mathcal{T}_r$ is a terminological base when the capability application on the heterogeneous environments, that we will mention in chapter 6.

We mentioned in this section the classification approaches on concepts and roles. These approaches are based on two couples of concepts : MSCs and LSCs, MSRs and LSRs. We saw that the subsumption relationship test is the basis for the calculation the two couples of concepts. So, we move now to the actual concept subsumption testing.

5.1.2 Subsumption Testing

A subsumption testing algorithm checks, given two concept descriptions, whether one can be embedded into the other [59]. We apply a Normalize-Compare algorithm (NC algorithm) for testing the subsumption relationships, that is a structural subsumption algorithm. Indeed, the algorithm compares the syntactic structures of the concept descriptions. We know that the structural subsumption algorithm cannot handle well the negation and the disjunction syntactic structures, but it is convenient for the calculation of the *complement* concept (see its definition in section 3.2.2) which is the formal basis for the composite answers we aim at providing.

The NC algorithm proceeds in two phases : normalization and comparison. First, the concept descriptions to be tested for subsumption are normalized, and then the syntactic structures of the normal forms are compared. These phases are successively discussed hereafter.

Normalization in $\mathcal{ALN}_{r+}$

Before going through the actual normalization process, we need an *atomization* preprocess in $\mathcal{ALN}_{r+}$ TBox. Indeed, as introduced in section 4.1, it exists two kinds of concept descriptions in $\mathcal{ALN}_{r+}$: *defined concept* and *primitive concept*.

A new *primitive concept* is introduced thanks to other *primitive concepts* or to the TOP concept into a $\mathcal{ALN}_{r+}$ TBox, and the new *primitive concept* is subsumed by the introducer.

A new *defined concept* can be introduced by a description form in $\mathcal{ALN}_{r+}$: *Concept identifier* $\doteq$ *concept definition*. The identifier of the new concept is equivalent to the description form of the definition in the $\mathcal{ALN}_{r+}$ TBox. Therefore, the *atomization* process *"unfolds"* the definition of the defined concept by recursively substituting concepts definitions to concept identifiers which appear in the right part of the concept definition (i.e. the right side of $\doteq$).

For example, consider the *defined concept* CITY-AIR-TRAIN, defined as :

CITY-AIR-TRAIN $\doteq$ (and CITY-AIRPORT
CITY-TRAIN).

On the right side of the definition symbol ($\doteq$), the concepts CITY-AIRPORT and CITY-TRAIN are defined concepts in the TBox $\mathcal{T}$ (see figure 4.6 on page 80). The definitions of the concepts CITY-AIRPORT and CITY-TRAIN in $\mathcal{T}$ are :

CITY-AIRPORT $\doteq$ (and CITY
(all has-flight* CITY)
(at-least 1 has-flight$^-$)),

CITY-TRAIN $\doteq$ (and CITY
(all has-train CITY)
(at-least 1 has-train$^-$)).

The atomization of CITY-AIR-TRAIN substitutes to CITY-AIRPORT and CITY-TRAIN their respective definitions, in the right part of the definition of CITY-AIR-TRAIN, resulting into :

CITY-AIR-TRAIN $\doteq$ (and CITY
(all has-flight* CITY)
(at-least 1 has-flight$^-$)
CITY
(all has-train CITY)
(at-least 1 has-train$^-$)).

The concepts, CITY, AIRPLANE, and TRAIN, are primitive concepts, so the atomized description form is written in primitive concepts. If it still exists a *defined concept* in the resulting description forms, the atomization process iterates, until all the concepts are primitive in the resulting description form. Normalization rules, N1 to N6 in figure 5.4, are then applied on the atomized form. In fact, normalization is coupled with simplification according to the normalization and the simplification functions for $\mathcal{ALN}_{r+}$ in figure 5.4, where these functions are denoted as Norm() and Simp(), respectively.

One should notice that the negation operator is only accepted in primitive concepts in $\mathcal{ALN}_{r+}$ (see for example, the rule N2, that eliminates some problem by using DE Morgan's law). For example, we would like to discover a direct contradiction between $(C \sqcap D)$ and $(\neg C \sqcup \neg D)$, but $\mathcal{ALN}_{r+}$ does not support disjunction operation between concepts.

Further, the normalization rules N5 and N6 incorporate the *at-least* and *at-most* restrictions into a single number restriction structure.

In addition, the normalization process calls a range of simplifications so that syntactically obvious contradictions and tautologies are eliminated. In the simplification rule S3, the $\sqcap S$ and $\sqcap P$ note the conjunction sets of primitive concepts. Back to the example of normalization on concept `CITY-AIR-TRAIN` in $\mathcal{T}$, the final result of the normalization process is :

$$\begin{aligned}\texttt{CITY-AIR-TRAIN} \doteq\ & (\texttt{and CITY}\\ & \quad (\texttt{all has-flight}^{*}\ \texttt{CITY})\\ & \quad (\infty\ 1\ \texttt{has-flight}^{-})\\ & \quad (\texttt{all has-train CITY})\\ & \quad (\infty\ 1\ \texttt{has-train}^{-})).\end{aligned}$$

A1 :	$\texttt{Atom}(C)$	=	definition form of C for defined concept name C
N1 :	$\texttt{Norm}(A)$	=	A for atomic concept name A
N2 :	$\texttt{Norm}(\neg A)$	=	$\texttt{Simp}(\neg\texttt{Norm}(A))$
N3 :	$\texttt{Norm}(C_1 \sqcap \cdots \sqcap C_n)$	=	$\texttt{Simp}(\sqcap\{\texttt{Norm}(C_1)\} \cup \cdots \cup \{\texttt{Norm}(C_n)\})$
N4 :	$\texttt{Norm}(\forall R.C)$	=	$\texttt{Simp}(\forall R.\texttt{Norm}(C))$
N5 :	$\texttt{Norm}(\geq n\ R)$	=	$\texttt{Simp}(\geq \infty\ n\ R)$
N6 :	$\texttt{Norm}(\leq n\ R)$	=	$\texttt{Simp}(\geq n\ 0\ R)$
S1 :	$\texttt{Simp}(A)$	=	A for atomic concept name A
S2 :	$\texttt{Simp}(\neg A)$	=	$\bot$, if $A = \top$ $\top$, if $A = \bot$ $\texttt{Simp}(B)$, if $A = \neg B$ $\neg A$, otherwise
S3 :	$\texttt{Simp}(\sqcap\mathbf{S})$	=	$\bot$, if $\bot \in \mathbf{S}$ $\bot$, if $\{A, \neg A\} \subseteq \mathbf{S}$ $\top$, if $\mathbf{S} = \emptyset$ $\texttt{Simp}(\mathbf{S} \setminus \{\top\})$, if $\top \in \mathbf{S}$ $\texttt{Simp}(\sqcap\mathbf{P} \cup \mathbf{S} \setminus \{\sqcap\{\mathbf{P}\}\})$, if $\sqcap\{\mathbf{P}\} in \mathbf{S}$ $\sqcap\mathbf{S}$, otherwise
S4 :	$\texttt{Simp}(\forall R.A)$	=	$\top$, if $A = \top$ $(\geq\ \infty\ 0\ R).A$, otherwise
S5 :	$\texttt{Simp}(\geq m\ n\ R)$	=	$\bot$, if $n \geq m$ $\geq m\ n\ R$, otherwise

FIG. 5.4 – Atomization, Normalization, and Simplification on $\mathcal{ALN}_{r+}$

Conceptually, a description in $\mathcal{ALN}_{r+}$ can be abstracted into the class diagram shown

in figure 5.5 and at the end of the normalization process, i.e. once the *normal form* of the concept description, as abstracted in figure 5.6[8], we enter the second step of the Normalize-and-Compare algorithm.

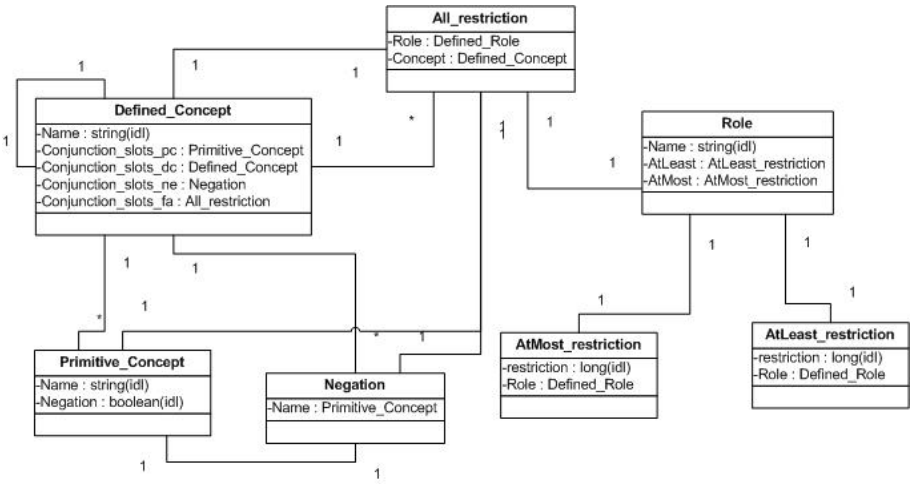


FIG. 5.5 – Description Form in $\mathcal{ALN}_{r+}$

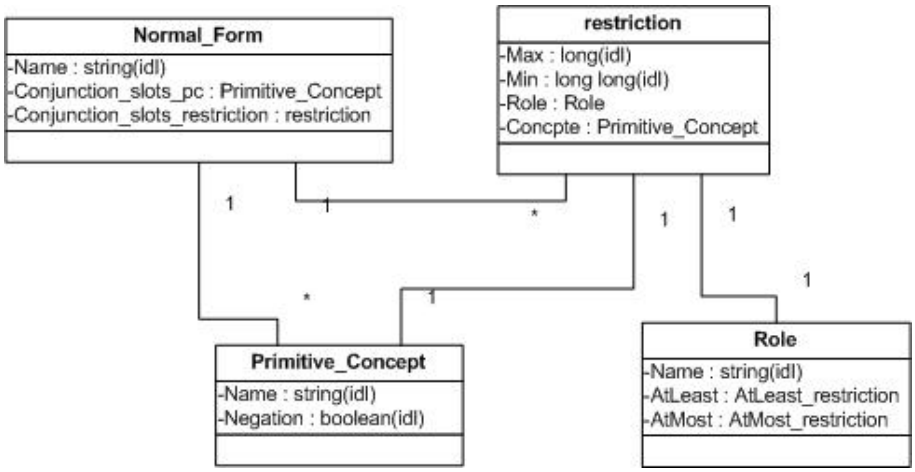


FIG. 5.6 – *Normal form* in UML

Normal Forms Comparison

Once normalized, two descriptions are easily compared against the subsumption relationship as explained hereafter.

Indeed, an $\mathcal{ALN}_{r+}$-concept description is in normal form iff it is under the form

$$\mathcal{F} = A_1 \sqcap \cdots \sqcap A_o \sqcap \neg A_{o+1} \sqcap \cdots \sqcap \neg A_{p-1} \sqcap m_p n_p R_p A_p \sqcap \ldots m_q n_q R_q A_q,$$

[8]From an implementation point of view, the normalization process reduces the class diagram in figure 5.5 into the one in figure 5.6

where $A_1, \ldots, A_q$ are distinct concept names, $R_p, \ldots, R_q$ are distinct role names.

It exists two kinds of atomic structures in the $\mathcal{ALN}_{r+}$ normal form concept description, which are the primitive concept A (or a negated primitive concept $\neg A$) and the number restriction $\geq m\ n\ R\ A$. According to the atomization, normalization and simplification rules in figure 5.4, any $\mathcal{ALN}_{r+}$ description can be transformed into a conjunction of atomic concepts, which is associative and commutative. So, the comparison process compares two normal forms according to the following rules.

Definition 5.1.5. *Let*

$$\mathcal{F}_C = A_1 \sqcap \cdots \sqcap A_o \sqcap \neg A_{o+1} \sqcap \cdots \sqcap \neg A_{p-1} \sqcap m_p n_p R_p A_p \sqcap \ldots m_q n_q R_q A_q$$

be the normal form of an $\mathcal{ALN}_{r+}$-concept description C, and

$$\mathcal{F}_D = B_1 \sqcap \cdots \sqcap B_u \sqcap \neg B_{u+1} \sqcap \cdots \sqcap \neg B_{v-1} \sqcap g_v k_v S_v B_v \sqcap \ldots g_w k_w S_w B_w$$

the normal form of an $\mathcal{ALN}_{r+}$-concept description D.

Then $C \sqsubseteq D$ iff the following two conditions hold :

1. *for all i in $\mathcal{F}_C$, $1 \leq i \leq p-1$, there exists j in $\mathcal{F}_D$, $1 \leq j \leq v-1$ such that $A_i = B_j$;*
2. *for all i $\mathcal{F}_C$, $p \leq i \leq q$, there exists j in $\mathcal{F}_D$, $v \leq j \leq w$ such that $A_i \sqsubseteq B_j$, and $R_i \sqsubseteq S_j$, and $m_i \geq g_j \geq k_j \geq n_i$.*

We will syntactically denote $\mathcal{ALN}_{r+}$ normal forms as simplified conjunctive forms. For example, A and B will be denoted as $A = (\text{ and } A_1, \ldots, A_n)$ and $B = (\text{ and } B_1, \ldots, B_m)$. Under these forms, two *concepts* can be easily compared to check whether the subsumption relationship holds between them or not :

giving $A = (\text{ and } A_1, \ldots, A_n)$ and $B = (\text{ and } B_1, \ldots, B_m)$, the test "does the concept A subsume the concept B ?" returns "true", if and only if $\forall A_i$ (i $\in$ 1,...,n) $\exists$ B_j (j $\in$ 1,...,m) such that $B_j \sqsubseteq A_i$.

For example, consider again the following concept descriptions of C and D.

```
(C)     CITY-AIRPORT  ≐  ( and CITY
                             (all has-flight CITY)
                             (at-least 1 has-flight⁻))

(D)  CITY-AIR-TRAIN   ≐  ( and CITY
                             (all has-flight CITY)
                             (at-least 1 has-flight⁻)
                             (all has-train CITY)
                             (at-least 1 has-train⁻))
```

Their normal forms are the following :

$$
\begin{array}{rll}
C: & (\sqcap\ \texttt{CITY} & (C_1) \\
 & \quad (\geq\ \infty\ 1\ \texttt{has-flight}^{-}.\texttt{CITY})) & (C_2) \\
 & & \\
D := & (\sqcap\ \texttt{CITY} & (D_1) \\
 & \quad (\geq\ \infty\ 1\ \texttt{has-flight}^{-}.\texttt{CITY}) & (D_2) \\
 & \quad (\geq\ \infty\ 1\ \texttt{has-train}^{-}.\texttt{CITY})) & (D_3)
\end{array}
$$

And the comparison of the two normal forms yields :

- C_1 and D_1 describe the same primitive *concepts* (CITY) ; therefore $D_1 \sqsubseteq C_1$ in accordance with the rule (1) of definition 5.1.5 (page 99) ;
- C_2 and D_2 are `number restriction` atomic structures, and then $D_2 \sqsubseteq C_2$ according to the rule (2) in the same definition.

Therefore, we can conclude that D is subsumed by C, i.e., CITY-AIR-TRAIN $\sqsubseteq$ CITY-AIRPORT.

5.1.3 Capability Management : Concluding Remarks

In this section, we presented the NC subsumption test algorithm in $\mathcal{ALN}_{r+}$. The four steps : atomization, normalization, simplification, and comparison process, ensure that this NC algorithm is sound and complete in $\mathcal{ALN}_{r+}$ TBox. The subsumption test is the foundation for concept and role classification and it is also the basis of all the kinds of reasoning we propose in this thesis work. We will use these characterizations to implement the subsumption test, the complement concept calculation, and the determination of composite answers in the coming section.

5.2 Capability Discovery

We introduced a conceptual model of *capability discovery* (section 2.3.3, page 57), where we sketched capability discovery and the related *concepts* (figure 3.2, page 62). We also introduced the notion of composite answer concept (section 3.2.2, page 66). The determination of a composite answer needs a kind of "measurable" satisfaction test method, that is why we choose to use the NC algorithm for the subsumption test algorithm. Indeed, the determination of a composite answer requires an algorithm which calculates the part of a query which is satisfied (in $\mathcal{L}_{satisfaction}$, according to our conceptual model), and the part which is missing (in $\mathcal{L}_{complement}$). The *satisfaction* and the *complement* are not complete *concept descriptions* in many situations, so we call them as a *capability description* in this thesis work.

Let us now introduce our proposals for the determination of a composite answer :

- Section 5.2.1 illustrate the way the normalize-and-compare algorithm is employed for evaluating queries ;
- Section 5.2.2 details how a composite answer can be built thanks to the complement concept ;

- Section 5.2.3 explains the principle of the query evaluation and the composite answer calculation in presence of multiple knowledge bases ;
- and finally, section 5.2.4 shows the individualization of the answer for a given query, i.e. the identification of the individuals that satisfy the query.

5.2.1 Composite Answer in $\mathcal{ALN}_{r+}$

A query action is a satisfiable inference calculation from a logical view, as mentioned in definition 3.2.2, page 67. A *subsumed concept* always satisfies the *subsumer concept* in $\mathcal{ALN}_{r+} - \mathcal{T}$, i.e., if the subsumer concept represents a query, its subsumed *concepts* will be a full satisfaction answer. Before determining a composite answer, we must calculate two basic components of composite answers, *satisfaction* and *complement* (We gave the definitions and a simple modeling descriptions of satisfaction and complement in section 3.2.2).

Basically, we use a concept description C to represent the requirement of a given query. If we can find a concept description D which satisfies $D \sqsubseteq C$, we can say that D is a full satisfaction for the query.

Let us come back to the subsumption test which was introduced in the previous section. The NC algorithm rewrites the concept descriptions into conjunctive forms before the comparison process. There are four kinds of atomic structures in the normal conjunctive form and we provided the rules of subsumption relationship test for the four kinds of atomic structures in definition 5.1.5. According to 2 rules of atomic structure subsumption relationship, we can simply see the normal form is a simple conjunctive form, as we have applied in the example of subsumption relationship test between `CITY-AIR-TRAIN` and `CITY-AIRPORT` in the previous section.

A concept description C can represent the requirement of a query Q as $C_1 \sqcap \cdots \sqcap C_n$ when we use a concept description C to represent the requirement. And then the possible answer will be the concept description D, which also is under a conjunctive form $D_1 \sqcap \cdots \sqcap D_n$. The conjunctive normal forms are then used to determine the *satisfaction* and the *complement*.

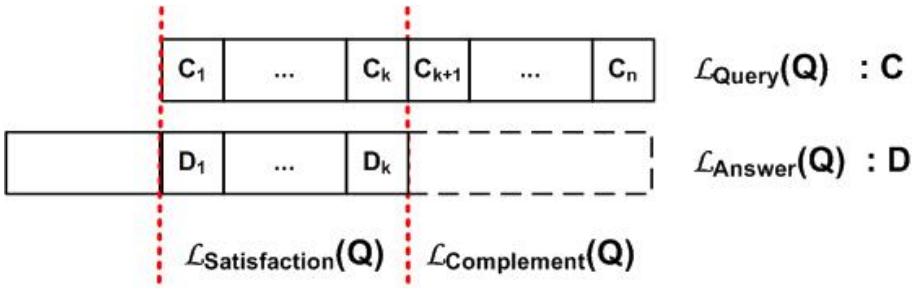


FIG. 5.7 – Determination of a Composite Answer

In the figure 5.7, C represents a query, and D represents a possible satisfaction, by the condition $1 \leq k \leq n$. In the concept descriptions C and D, the atomic structures $C_1, \ldots, C_k$ and $D_1, \ldots, D_k$ meet the comparison conditions in the definition 5.1.5

(page 99). So the satisfaction description is : $\mathcal{L}_{Satisfaction}(Q) = D_1 \sqcap \cdots \sqcap D_k$.
The atomic structures $C_{k+1}, \ldots, C_n$ are not satisfied, and they are in the complement description : $\mathcal{L}_{Complement}(Q) = C_{k+1} \sqcap \cdots \sqcap C_n$.

As an illustration, let us try to determine the composite answer, that uses satisfaction from the concept description $D \doteq$ `CITY-AIRPORT`, for the query description $C \doteq$ `PORT-AIR`. The corresponding normal form descriptions are as follows :

$$
\begin{array}{rll}
C: \texttt{PORT-AIR} = & (\sqcap\ \texttt{CITY} & (C_1) \\
& \quad (\geq\ \infty\ 1\ \texttt{has-flight}^{-}).\texttt{CITY}) & (C_2) \\
& \quad (\geq\ \infty\ 1\ \texttt{has-ferry}).\texttt{CITY})) & (C_3) \\
 & & \\
D: \texttt{CITY-AIRPORT} = & (\sqcap\ \texttt{CITY} & (D_1) \\
& \quad (\geq\ \infty\ 1\ \texttt{has-flight}^{-}).\texttt{CITY})) & (D_2)
\end{array}
$$

In the concept descriptions C and D, the atomic structures C_1, C_2 and D_1, D_2 meet the conditions in the definition 5.1.5, so we get the partial satisfaction description $D_1 \sqcap D_2$. However, the atomic structure C_3 is not satisfied, so we get the complement description C_3. According to the *Ask* action (as defined in definition 3.1.2 on page 61), we have :

$$
\begin{array}{rrl}
\text{Ask}: & \mathcal{L}_{Query}(Q) & = \texttt{PORT-AIR} \\
 & \times & \\
 & KB & = \{\top, \bot, \texttt{CITY-AIRPORT}\} \\
 & \rightarrow & \\
 & \mathcal{L}_{Answer}(Q) & = \{\mathcal{L}_{Satisfaction}(Q) = (D_1 \sqcap D_2, \\
 & & \quad \mathcal{L}_{Complement}(Q) = (C_3)\}
\end{array}
$$

where C_i or D_i do not necessarily correspond to any $\mathcal{ALN}_{r+}$ concept description in $\mathcal{T}$. These two kinds of atomic description structures are the minimal unit in the determination of a composite answer and following the capability discovery approach. Of course, the satisfaction and the complement, which are under the conjunctive form of atomic description structures, also do not necessarily correspond to any $\mathcal{ALN}_{r+}$-concept description in $\mathcal{T}$. We name this conjunction of atomic description a *capability description.*

5.2.2 Query Complement Determination

The determination of a query complement proceeds in two steps :

1. Identify the complement concept, i.e. the part of the query that is not satisfied ;
2. Actually find the complement, if it exists, in the knowledge base.

These steps are successively detailed hereafter.

Complement Identification

We mentioned that multiple satisfactions may exist and that they have to be composed to satisfy the requirement of a query (section 3.2, page 65). In order to calculate the satisfaction and the complement, we introduce an array of Boolean (called Satisfaction Table further, ST for short). The Boolean array records the results of the evaluation of the subsumption relationship (see table 5.1). In that figure, $C_1, C_2, C_3, \ldots, C_n$ denote the query concept under its normal form and $D_1, D_2, D_3, \ldots, D_m$ denotes *concepts* in $\mathcal{T}$ under a normal form too, i.e. every D_j has to be viewed as $(and D_j^1, D_j^2, \ldots, D_j^{nj})$. Therefore,

$$ST[D_j, C_i] = true \leftrightarrow \ D_j^i \sqsubseteq C_i$$

For implementation purposes, a function *Subsumes(Subsumer, Subsumed)* checks whether the subsumption relationship holds between *Subsumer* and *Subsumed*. When the value returned by the function $Subsumes(C, D_j)$ is "false" (i.e. the concept D_j does not fully satisfy the concept C), we need to determine a possible complement of D_j relatively to C, using the satisfaction table.

In the example about `PORT-AIR` and `CITY-AIRPORT`, in the previous section,

$$Subsumes(\texttt{PORT-AIR}, \texttt{CITY-AIRPORT}) = \texttt{false},$$

i.e. `CITY-AIRPORT`$\neg \sqsubseteq$ `PORT-AIR`, while `CITY-AIRPORT` does not fully satisfy the `PORT-AIR`. "($\geq \ \infty$ `1 has-ferry`).CITY" is the unsatisfied part. Then the complement is

$$Comp(\texttt{PORT-AIR},\texttt{CITY-AIRPORT}) = (\geq \ \infty \ \texttt{1 has-ferry}).\text{CITY}).$$

More generally, using the satisfaction table, the complement (that remains to be satisfied), denoted as $Comp(Query, Satisfaction)$ satisfies the following condition :

Definition 5.2.1. $Comp(C, D) = \sqcap_{k=1}^{r} C_k \mid \forall k \ ST[k] = false.$

That means that the *complement* is given by the conjunction of all the *atomic concepts* for which the corresponding values in the satisfaction table are "false".

	C_1	C_2	...	C_n
D_1	False	False	...	True
D_2	False	True	...	True
...	...	...	...	...
D_m	False	False	...	False
ORoD	False	True	...	True

ORoS	ANDoS
True	False

TAB. 5.1 – Three parameters of the satisfaction table : ORoD, ORoS and ANDoS

Complement Calculation

Let us first explain the principles of the actual calculation of the complement, before generalizing it into an algorithm.

a) Complement Calculation Principles and Examples : In order to attempt to actually satisfy the identified complement, we introduce three boolean variables, as parameters of satisfaction table : ORoD, ORoS, and ANDoS (refer again to table 5.1). The three parameters are used to determine the cases of satisfaction, that is the cases of satisfaction that have been introduced in section 3.2 (page 65) and that are re-called here :

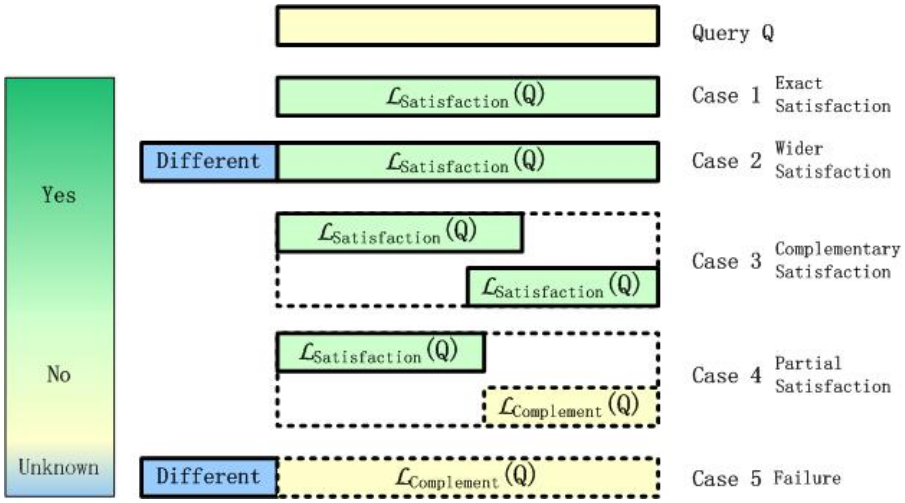


FIG. 5.8 – Query Satisfaction Situations

$ORoD[1..n]$ is defined as $ORoD[i] = \bigvee_{j=1}^{m} ST[D_j^i, C_i], \forall i \in [1..n]$.

$ORoD[i] = true$ means that the concept C_i is satisfied by at least one $D_k^i, k \in [1..m]$.

A similar way, if the conjunction of the values of $ORoD$, denoted $ANDoS$ and defined as $ANDoS = \bigwedge_{i=1}^{n} ORoD[i]$, is true, then that means that all the C_is are satisfied, and therefore the query is satisfied too.

Finally, when $ANDoS$ is false, the logical disjunction of the values of $ORoD$, noted $ORoS$ and defined as $ORoS = \bigvee_{i=1}^{n} ORoD[i]$, enables to determine a possible partial satisfaction.

Indeed, if $ORoS = True$, that means that there exists some C_k that are satisfied. If both $ORoS$ and $ANDoS$ are false then no atomic concept description D_j^k $(j \in 1..m)$ satisfies any C_i.

Thanks to the analysis of the results of the classification (which identifies *Least Common Subsumer (LCS)* and *Most Specific Concept (MSC)*) and the results of the NC algorithm, which delivers the ORoS and ANDoS values, the cases for a composite answer are covered : Table 5.2 summarizes this discussion. In this table, X and Y are two concept descriptions, $\top$ is the TOP concept and $\bot$ is the BOTTOM concept.

LCS(Q)	MSC(Q)	ORoS	ANDoS	CASE
X	Y	True	True	1 : Exact Satisfaction
X	$\bot$	True	True	2 : Wider Satisfaction
$\top$	$\bot$	True	True	3 : Complementary Satisfaction
$\top$	$\bot$	True	False	4 : Partial Satisfaction
$\top$	$\bot$	False	False	5 : Failure

TAB. 5.2 – Analysis of the Satisfaction Cases

	C_1	C_2	C_3
D	True	True	False
ORoD	True	True	False

ORoS	ANDoS
True	False

FIG. 5.9 – Satisfaction Table : First Example

The numbers in the CASE column refers to the satisfaction cases in figure 5.8. We can express the five satisfaction cases decision rules as :

1. *Exact satisfaction* : $Y \sqsubseteq Q \sqsubseteq X$ iff $(LCS(Q) = X) \wedge (MSC(Q) = Y) \wedge (ORoS = True) \wedge (ANDoS = True)$;
2. *Wider Satisfaction* : $Q \sqsubseteq X$ iff $(LCS(Q) = X) \wedge (MSC(Q) = \bot) \wedge (ORoS = True) \wedge (ANDoS = True)$;
3. *Complementary Satisfaction* : $\exists X_1, \ldots, \exists X_n, Q \sqsubseteq \bigcup_{i=1}^{n} X_i$ iff $(LCS(Q) = \top) \wedge (MSC(Q) = \bot) \wedge (ORoS = True) \wedge (ANDoS = True)$;
4. *Partial Satisfaction* : $\exists X_1, \ldots, \exists X_n, \bigcup_{i=1}^{n} X_i \sqsubset Q \wedge \neg(\exists Y_1, \ldots, \exists Y_m, Q \sqsubseteq \bigcup_{j=1}^{m} Y_j)$ iff $(LCS(Q) = \top) \wedge (MSC(Q) = \bot) \wedge (ORoS = True) \wedge (ANDoS = False)$;
5. *Failure* : $\forall X \in \mathcal{T}, X \cap Q = \emptyset$ iff $(LCS(Q) = \top) \wedge (MSC(Q) = \bot) \wedge (ORoS = False) \wedge (ANDoS = False)$.

Back to the driving example, the values concerning $\mathcal{L}_{Query}$(`CITY-AIRPORT`) and $\mathcal{L}_{Satisfaction}$(`CITY-AIRPORT`) = {`PORT-AIR`}, are as follows :

ORoS = True and ANDoS = False, accords with the fourth line of the table 5.2, so the case of satisfaction is 4, i.e., `CITY-AIRPORT` partly satisfies `PORT-AIR`.

Now we suppose that a new concept description, E = `PORT`, is added to the knowledge base.

```
PORT=  (⊓ CITY                                  E1
          ((≥  ∞ 1 has-ferry).CITY))            E2
```

	C_1	C_2	C_3
D	True	True	False
E	True	False	True
ORoD	True	True	True

ORoS	ANDoS
True	True

FIG. 5.10 – Satisfaction Table : 2nd Example

PORT is inserted into the knowledge base KB by the *Tell* operation :

$$\begin{array}{rrl} \text{Tell}: & \mathcal{L}_{Tell}(E) & = \texttt{PORT} \\ & \times & \\ & KB & = \{\top, \bot, \texttt{CITY-AIRPORT}\} \\ & \rightarrow & \\ & KB' & = \{\top, \bot, \texttt{CITY-AIRPORT}, \texttt{PORT}\} \end{array}$$

We apply the concept description E as the satisfaction to the previous example $\mathcal{L}_{Query}(\texttt{CITY-AIRPORT})$ and $\mathcal{L}_{Satisfaction}(\texttt{CITY-AIRPORT}) = \{\texttt{PORT-AIR}, \texttt{PORT}\}$, and then recalculate the consulting values as follows.

Now the result ORoS = True and ANDoS = True, accords with the third line of the table 5.2, the situation of this answer is case 3 (Complementary Satisfaction), i.e. $\texttt{CITY-AIRPORT} \sqcap \texttt{PORT}$, which is a composite answer that fully satisfies the query $Q = \texttt{PORT-AIR}$.

b) Complement Calculation Algorithm : We review the previous example showing the important role played by the *satisfaction table*. In the first *satisfaction table* (figure 5.9), only one entity description D is in the satisfaction table, and the entity D does not fully satisfy the query description C. In the second example (figure 5.10), an other entity description is added in the *satisfaction table* that renders ORoS = True and ANDoS = True, i.e., the composition of (D and E) fully satisfy the query description C.

From an implementation perspective, in a *satisfaction table*, a description under its normal form is represented as an array of boolean variables where :

1. The size of the array is the number of atomic concept descriptions in the query concept description. In the figure 5.10, the query C is defined thanks to three atomic concept descriptions : therefore the size of the array is 3.
2. The values of the elements in the array reflect which atomic concepts (under the same subscript as the respective elements) are satisfied or not, according to the result provided by the subsumption testing algorithm. For example, the table entry of the entity D is {True, True, False}. The values of the first and the second elements being True, that means that the first and the second atomic concepts in C are satisfied by the concept D, while the third atomic concept in C is not satisfied.

In this example, there are only two entity descriptions (D and E), and the two entities "together luckily" fully satisfy the query. In fact, we may get lot of candidate entities descriptions from one or several knowledge bases. The choice among the candidates is a matter of search strategies that may change the algorithm behavior (as already discussed in section 3.2.2 on page 66).

Indeed, all the candidate entity descriptions are sorted according to a given criteria (also called *priority* hereafter) in order to help in deciding which entity description is to be considered at first in the *satisfaction table*. The priority criteria may be a simple one, like considering the concept with the larger number of satisfied atomic concepts satisfaction. But it may be a complex rule like *(i)* "Firstly, homogeneous concepts first and secondly, the largest satisfied concepts" or *(ii)* "The atomic concept C_1 and C_3 must be satisfied by a single entity", etc.

The sort order is the first step to get a composite answer, then we apply the entities candidates in the *satisfaction table* one by one. The ORoD and ANDoS will be recalculated, when a new entity candidate description is inserted in the *satisfaction table*.

- Firstly, if there are some variables in ORoD whose values is changed from False to True, then this entity is valuable for satisfying the given query description.
- Secondly, if ANDoS is True, then we find a full satisfaction for the query. We try all the possible combinations of entities, then we calculate the ORoD, ANDoS.

The abstract algorithm can be implemented in many different ways : one of its implementation is sketched in figure 5.11.

There are two objects, `Entities[1..n]` and `ST`, in the flowchart of the algorithm :

1. `Entities[1..n]` is a set objects of the candidate entities described in the array of boolean variables. We do not detail the sort order process of `Entities[1..n]`, because, as said before, the entity selection strategy can be under many forms. Actually, we implemented *"the highest number of satisfied concepts"* ordering.
2. The object `ST` holds a *Satisfaction Table* and several operations :
   - `ST.clear` creates and initializes the *Satisfaction Table* ;
   - `ST.add()` adds a new line at the end of the *Satisfaction Table* ;
   - `ST.DeleteLastLine` delete the last line of the *Satisfaction Table*.

All the three operations recalculate the `ORoD`, `ORoS` and `ANDoS`, and if any value in `ORoD` is changed, then the variable `ORoD_change` will be set at True, otherwise it is set at False.

With *"the highest number of satisfied concepts"* ordering strategy, the algorithm in figure 5.11 implements the *satisfaction first* strategy which was mentioned in section 3.2.2 (page 66). Changing the strategy requires changing the implementation of a decision box, as annotated in figure 5.11. For example, simply adding a limit `(and ST.index<2)` into the decision box will implement the *at most 2 answers* strategy, limiting the number of lines in the *Satisfaction Table* to at most 2.

This algorithm is working on these simple strategies, and finally it finds the entities which can satisfy the given query.

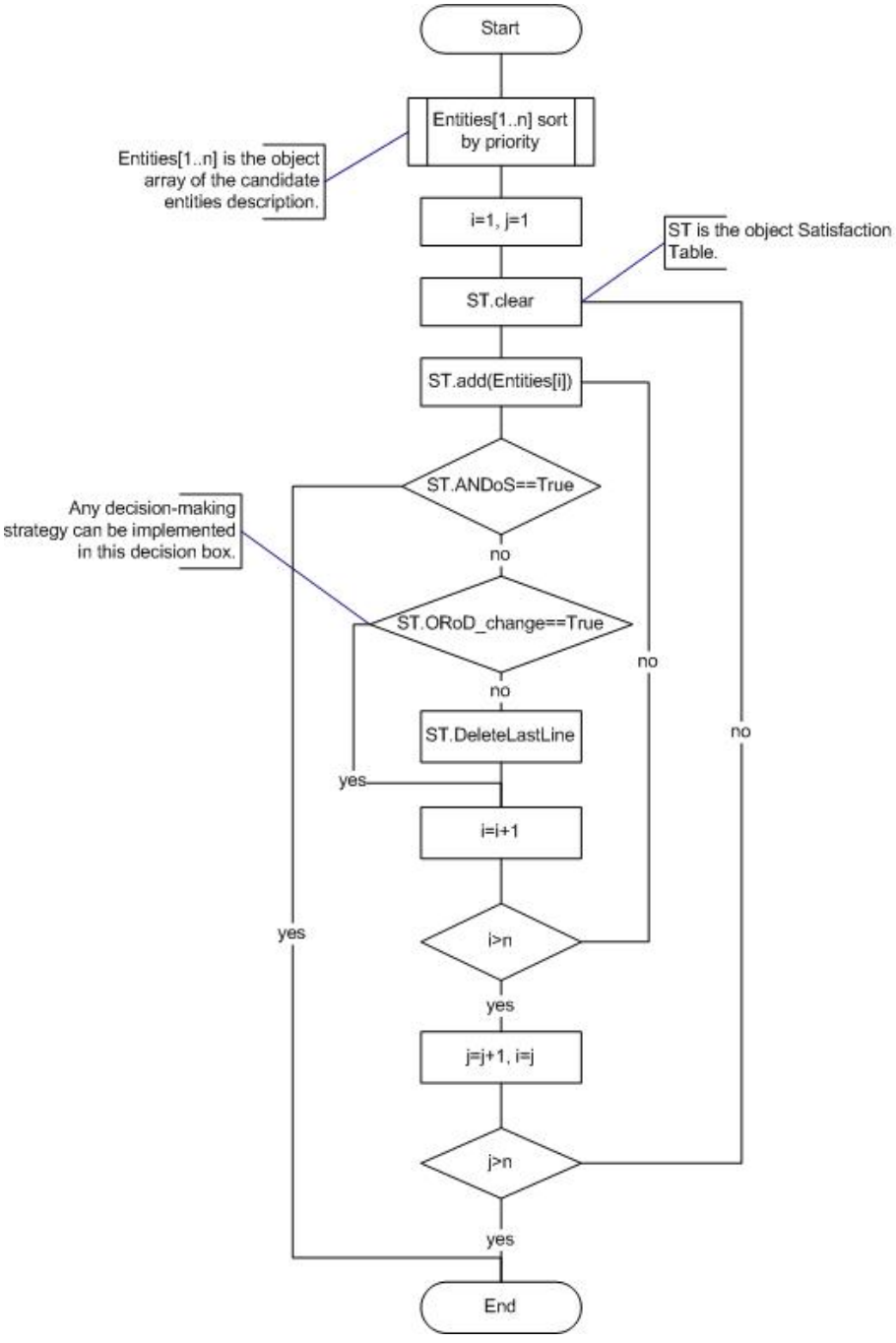


FIG. 5.11 – The Flowchart of the Algorithm on *Satisfaction Table*

Concluding Remarks

We introduced a complete method to determine the satisfaction cases that were listed in section 3.2.2. As we said in section 3.2.2, if the composite answer is under the cases four or five ("Partial satisfaction" or "Failure" within a single mediator), then we will send the *complement* description to other knowledge bases in the mediator federation. This topic is considered in the coming section.

5.2.3 Capability Discovery in Multiple Mediators

In the introductory chapter (section 1.2.2, page 32), we distinguished three types of discovery :

1. *Homogeneous local satisfaction*, where a single mediator satisfies a request,
2. *Homogeneous distributed satisfaction*, where multiple mediators, having the same Knowledge description language, complement each other to satisfy a request,
3. *Heterogeneous distributed satisfaction*, where multiple mediators having different Knowledge description languages have to cooperate in order to satisfy a query.

The previous section dealt with the first type of discovery and this section covers the two latter cases, a distinguishing feature among them being the fact that language translation mechanisms are required when the Knowledge representation languages are heterogeneous. Nevertheless, an additional mechanism may be needed in both the situations when the multiple mediators do not use a common vocabulary in their respective knowledge bases : in this situation, matching between terms and term translation are required. Indeed, in this kind of heterogeneous context, mapping of terms and concepts is often required between the cooperating mediators. The translation at the *syntax level* is applied to atomic capability descriptions which are accepted in different knowledge representation languages (see figure 2.3 on page 48) and when the vocabularies which are used by the mediators are different, a *lexical translation* is also performed.

These considerations are exposed in this section in the following way :

1. explain the notion and present the accepted forms of atomic capability descriptions ;
2. introduce the lexical translation of terms we have adopted ;
3. present the notion of composite answer in a heterogeneous environment ;
4. introduce the complement calculation process in this type of environment.

Atomic capability Description

As explained before, a query description is divided into a sequence of atomic concept descriptions. There are two kinds of atomic concepts in the normal form, which are the primitive concept A and the restricted concept $(m\ n\ R).A$. The primitive concepts usually define the basic terms in the knowledge base, and are indescribable. i.e., it is an entity which and only which own one characteristic/capability. So it exists one and only one capability description for each primitive concept in the $\mathcal{ALN}_{r+}$ T$_r$Box (If $A \in \mathcal{T}$, then it exists $R_A \in \mathcal{T}_r$.). The primitive concept can be written as the restricted structure,

$$\text{Universalization rule 1} : A \equiv \forall R_A.A.$$

For the second kind of atomic concept, the universal quantification structures are translated into a number-restricted structure thanks to the simplification rule S4 (see figure 5.4 on page 97). i.e., when $m = \infty$ and $n = 0$, the number-restricted structure is equivalent to the universal quantification structure,

$$\text{Universalization rule 2} : (\geq \ \infty \ 0 \ R).A \equiv \forall R.A.$$

The infinity (∞) is the maximal value of m, and the zero (0) is the minimal value of n, so the universal quantification structure subsumes the number restricted structure in $\mathcal{ALN}_{r+} : (\geq \ m \ n \ R).A \sqsubseteq \forall R.A$.

On the other hand, a composite answer accepts the partly satisfied answer as the satisfaction of the case 4 (*Partial satisfaction*). The most specific concepts fully satisfy the given query concept, and the least subsumer partly satisfies the query concept. The least subsumer will be the candidate in the satisfaction table, when a most specific concept does not exist. So, the universal quantification structure partly/fully satisfies the number-restricted structure.

From an other side, number-restricted structures are not accepted in many knowledge representation languages. We introduce a solution on capability discovery in the heterogeneous environment, so the description model must be supported in all knowledge representation language : we translate the normal form into a conjunction structure of universally quantified concept, where only one atomic concept structure exists : $\forall R.A$.

The structure of atomic concepts are one by one satisfied in the satisfaction table in the previous section. The unsatisfied structures in the *complement* may be satisfied by other systems.

The atomic concept structure ($\forall R.A$) is written in three symbols : universal quantification $\forall$, *Role* identity R, and *Concept* identity A, that means that all assertions of an entity A own the capability R. If the atomic concept structure represents a query $\mathcal{L}_{Query}(Q) = \forall R.A$, it says that we are looking for a entity A which owns the capability R, where A is the identity of one entity and the capability R is the one which is looked for.

Term translation and term matching

In a capability discovery query $\mathcal{L}_{Query}(Q) = \forall R.A$, the concept description A represents any entity which holds the capability R. So the concept description is translated into a *universal concept*, like the *top concept* $\top$ in DLs, and the *"thing"* concept in F-Logic and CGs. The vocabulary of R must be translated between the heterogeneous systems. The lexical translation is independent of the syntax translation. The terms of R are some identifiers of the capability in the query for capability discovery.

In a distributed environments, term mapping may be facilitated by the use of a common ontology. Approaches like [4, 74] use a common ontology and evaluate the semantic relations amount concepts. The most basic method simply computes the "distance" between two concepts in the common ontology. In our work, we introduce the capability description structure ($\forall R.A$) for the query between the heterogeneous systems. The vocabularies in the *capability description* must be included in the common ontology, which

is the $\mathcal{T}_r$ in this work. As discussed in our motivation of this work, we consider the *capability description* vocabulary as being a set whicj is set than the *concept description* vocabulary (see chapter 1.1. So it might be possible to manually build a common capability description ontology on a specific domain, as we did concerning the example about travels.

The manual common ontology is basic and it supports an "auto-learning" approach when an unknown term is introduced as a *capability description*. This auto-learning approach is implemented thanks to hierarchical dictionaries, like WordNet [65], and EuroWordNet [84]. However, classicaly, we need to tackle situations where exact mappings between terms do not exist. We dealt with such situations thanks to similarity distance measurement among terms.

There are multiple algorithms that measure the semantic similarity in a concept hierarchy [110, 82]. In this work, we opted for an algorithm based on the Least Common Subsumer (LCS) [110] to calculate the similarity between two terms. The similarity measurement function we used is defined as :

$$Sim(X,Y) = \frac{2\times|LCS(X,Y)|}{|X|+|Y|},\ 0 \leq Sim(X,Y) \leq 1,$$

X and Y being two terms in the hierarchical dictionary, $\mid X \mid$ is the shortest distance from the root node to the node of the term X in the taxonomy where the term of X lies. $LCS(X,Y)$ denotes the least common subsumer of X and Y and the more the value of $Sim(X,Y)$ is high (i.e. close to 1), the more the two terms X and Y are similar.

Therefore, a capability discovery query may include a *capability description* X which is not defined in the $\mathcal{T}_r$ box (which plays the role of the common ontology), but the term X exists in the hierarchical dictionary. Of course, all the terms in $\mathcal{T}_r$ are assumed to already exist in the hierarchical dictionary when we build it. We calculate the similarity value of the term X with each term in $\mathcal{T}_r$ and we retain the term(s) in $\mathcal{T}_r$ which is/are the most similar to X, i.e. the terms with the highest similarity degree.

Actually, this method of similarity measurement is achieved thanks to a hierarchical dictionary like WordNet[65] which provides the largest vocabulary, EuroWordNet[84] which supports multiple European languages or SNOMED-CT [82] which provides a specific vocabulary on medical domain (we can also find many different hierarchical dictionaries on many different specific domains). Theretofore, we consider that this approach is applicable to many domains thanks to the choice of the appropriate dictionary.

Capability Discovery and Heterogeneous Representations

In the capability discovery view, the approach for the determination of a composite answer always tries to maximally satisfy the capability description $\mathcal{L}_{Query}(Q)$ or $\mathcal{L}_{Complement}(Q)$. We introduced two kinds of operations : *conceptualization* and *individualization*, which translates the *capability discovery* into a *concept discovery* or an *individual discovery*, respectively (see figure 3.2, on page 62).

We recall that an ABox $\mathcal{A}$ contains two kinds of assertions of the form $C(a)$ and role assertions of the form $R(a,b)$, i.e. *individualization* exists in two conditions : concept-individual and role-individual. In a simplified view, the *individual discovery* can be seen

alike searching for an instance in a relational database with only unary or binary relations (We gave examples in section 4.2).

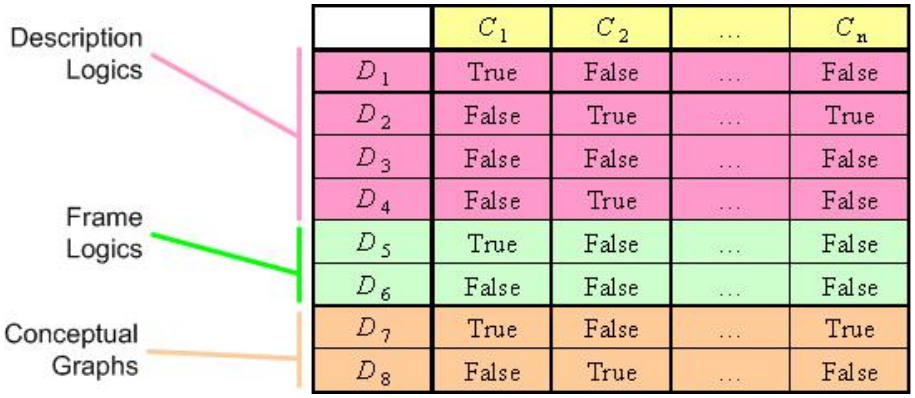
The *conceptualization* can transform the *capability discovery* into a *concept discovery*, as admitted by almost all knowledge representation languages. For example, the *conceptualization* easily writes a query of role R in the three logic languages we considered :

$$\begin{aligned} \mathcal{Q}_{DL} &\doteq \forall R.X\,; \\ \mathcal{Q}_{F-Logic} &\doteq \text{FORALL } X \leftarrow R(X)\,; \\ \mathcal{Q}_{CGs} &\doteq [\text{Object} :^{*}X]\ [\text{Object} :Y]\ (R\ ?X\ ?Y)\,; \end{aligned}$$

And then this query description is easily transformable into any knowledge description (see the analysis table in figure 2.3, page 48). Therefore, we can process this query alike the normal concept discovery.

This principle helps us to apply the composite answer approach in a heterogeneous knowledge representation environment. As said before, in this work, we considered, for illustration purposes, three kinds of knowledge representation technologies : Description Logics, Frame Logics, and Conceptual Graph ; And as analyzed in the table 2.3 (on page 48), the capability description $\mathcal{L}_{complement}(Q)$ can be accepted in the three kinds of KR languages.

In this context, the satisfaction table for the given heterogeneous KR is illustrated by the figure 5.12.



	C_1	C_2	...	C_n
D_1	True	False	...	False
D_2	False	True	...	True
D_3	False	False	...	False
D_4	False	True	...	False
D_5	True	False	...	False
D_6	False	False	...	False
D_7	True	False	...	True
D_8	False	True	...	False

FIG. 5.12 – Composite Answer in Heterogeneous KR

In that figure,

- $C_1, \ldots, C_n$ is the normal form of a query Q,
- $D_1, \ldots, D_8$ are the $\mathcal{L}_{Satisfaction}(Q)$ coming from different knowledge representation systems.

In addition, as an example, we assume that D_1, D_2, D_3, D_4 are in Description Logics, D_5 and D_6 are in Frame Logics and, D_7 and D_8 are in Conceptual Graphs.

The algorithms and the approach we applied in a homogeneous environment can be easily applied in a heterogeneous one : a *composite answer* model was illustrated in the example that used the concepts `PORT-AIR` and `CITY-AIRPORT`, in section 5.2.1, where, the

CITY-AIRPORT concept partly satisfies the PORT-AIR concept. And then we gave a homogeneous composite answer adding a new concept description PORT into the TBox $\mathcal{T}$. Then the composite answer (PORT⊓ CITY-AIRPORT) fully satisfied the concept PORT-AIR. Let us elaborate more on the complement calculation when faced to heterogeneous representations.

Complement Calculation within Heterogeneous Representations

As shown by the example, some queries may not be fully satisfied within a single knowledge base. A current solution is to ceaselessly increase the knowledge base to increase the quality of a query/answer service, as we did in the previous section. But the huge knowledge base needs more and more memory space, CPU time, etc., that makes the system spending more time and resource for processing one query/answer service task. Finally, it is not reasonable nor feasible to build a universal knowledge base which "knows everything". In fact, the knowledge which is missing in a local knowledge base may exist in an other one, and possibly in an other knowledge representation language.

The capability discovery approach we are proposing can use distributed and heterogeneous knowledge (Considerations about distributed knowledge system research were introduced in section 2.3). In our composite answer approach, we send the *complement* as a query $\mathcal{Q}_{\mathcal{T}2}$, $\mathcal{Q}_{\mathcal{T}2} = Comp(Q, \mathcal{T}_1)$, to an other mediator server. The results returned by the other mediator are put in the satisfaction table for possibly determining a composite answer as in the previous section. We focus on the problem of heterogeneous knowledge representations in this section.

In the continuation of the example on PORT-AIR and CITY-AIRPORT in section 5.2.1, the concept CITY-AIRPORT partly satisfies the PORT-AIR concept, and the complement in this DL knowledge base TBox is

$$Comp(\texttt{PORT-AIR}, \mathcal{T}_{DL}) = ((\geq \ \infty \ \texttt{1 has-ferry}).\text{CITY}).$$

We calculate the consulting values on $\mathcal{L}_{Query}(\texttt{CITY-AIRPORT})$ and we get : $\mathcal{L}_{Satisfaction}(\texttt{CITY-AIRPORT}) = \{\texttt{PORT-AIR}\}$. Therefore, the expresion of the results in the satisfaction table is as follows.

	C_1	C_2	C_3
D	True	True	False
ORoD	True	True	False

ORoS	ANDoS
True	False

ORoS = True and ANDoS = False, accords with the fourth line of the table 5.2, so the case of satisfaction is 4 (Partial satisfaction), i.e., CITY-AIRPORT partly satisfies PORT-AIR.

Now suppose that an other mediator server exists in Frame Logic. The F-Logic mediator only accepts query descriptions in F-Logic. Applying the universalization rule 2, we rewrite $\mathcal{Q}$, $\mathcal{Q} = \forall \texttt{has-ferry}.X$, in F-Logic as $\mathcal{L}_{F-Logic}(Q) =$ (FORALL $X \leftarrow$ $\texttt{has-ferry}(X)$)

Let the concept $E_{\mathcal{T}_{F-Logic}}$ identify the F-Logic mediator. If $E_{\mathcal{T}_{F-Logic}}$ returns a result (other than a failure), then we insert *True* in the satisfaction table, resulting therefore into the following table :

	C_1	C_2	C_3
$D_{\mathcal{T}_{DL}}$	True	True	False
$E_{\mathcal{T}_{F-Logic}}$	False	False	True
ORoD	True	True	True

ORoS	ANDoS
True	True

Now, ORoS = True and ANDoS = True conforms to the third decision line of the table 5.2, the situation of this answer is case 3 (complementary satisfaction), i.e., the composite answer $D_{\mathcal{T}_{DL}} \sqcap E_{\mathcal{T}_{F-Logic}}$ fully satisfies the requirement of the query Q=`PORT-AIR`. In this example, the query Q is represented in DL, so DL language is the local knowledge representation language. In the composite answer, the concept description $D_{\mathcal{T}_{DL}}$ is in $\mathcal{ALN}_{r+}$, which is the same language as the language of the initial query Q ; the concept description $E_{\mathcal{T}_{F-Logic}}$ is in F-Logic which is a language different from the language of the query Q. So the composite answer is composed in two heterogeneous knowledge representation languages. This approach can be extended to other environments, as shown for environments including DLs and CGs, for example.

Conclusion

In this section, we introduced the composite answer approach in the heterogeneous environments. All the approaches offer the composition of concepts for a given query. Finally, the result must be individualized in order to actually provide a service. We introduce the individualization of the capability discovery in the next section.

5.2.4 Conceptualization and Individualization

From an application point view, capability management and discovery approaches must serve some valuable tasks. Alike a Relational Data Base Management System (RDBMS), where we create tables to save the data, and we create views to find data for a given condition, capability discovery must finally return one or several entities/individuals which satisfy the requirements of a given query. This section details the conceptualization approach and the individualization approach in a *capability discovery* system.

A *capability discovery* is performed thanks to an *atomic concept* discovery. We sketched the relationship between *capability discovery* and *concept discovery* in figure 3.2 (page 62), where the *concept discovery* is called *model discovery*. The conceptual model of capability discovery can be implemented in $\mathcal{ALN}_{r+}$ based on the methods which were introduced in section 5.1. We represent hereafter the implementation of the capability discovery conceptual model in $\mathcal{ALN}_{r+}$-system, as shown in figure 5.13.

The capability discovery approach includes two kinds of implementations : conceptualization and individualization. The *individualization process* is similar to the SQL-*select* query in a RDBMS : we know the name of the *concept* then we can find which individuals are under the concept, quite alike knowing the name a relational table and finding all data in that table.

A similar way, we can find which individuals hold a capability when we know the name of a *role* in the $\mathcal{ALN}_{r+}$-system (This individualization process can also be implemented

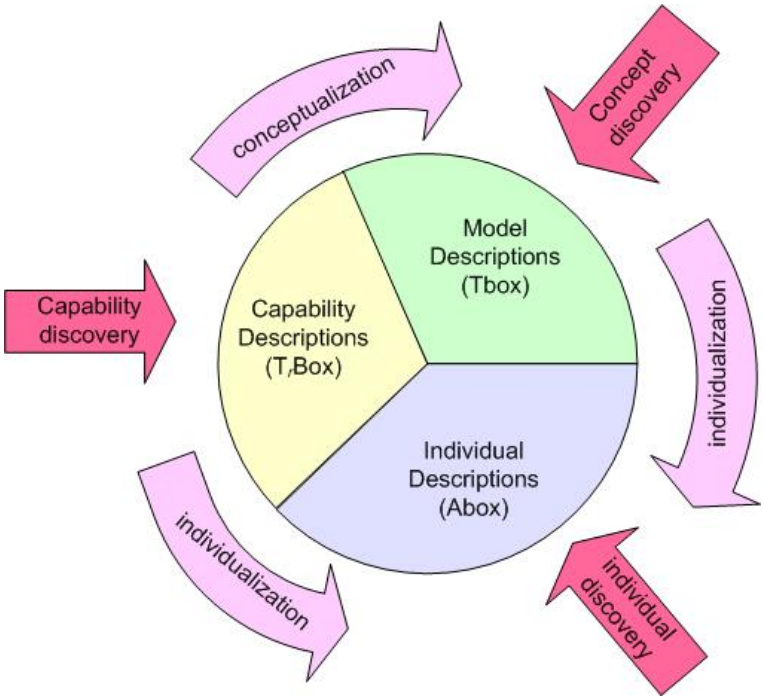


FIG. 5.13 – Capability discovery approach in $\mathcal{ALN}_{r+}$-system

in any relational data system : we create data tables for each role and each concept, then we save the individual's information in the corresponding tables when an individual is under a given concepts, or holds a given capability. Further, the individualization can be implemented by the API of current application system, which support a process like SQL-*select* statement. We give a term description, then the system can give all the individuals' information under that term. Anyway, the individualization finally carries out the *capability discovery*, which finds the individuals who hold a capability).

In fact, the *individualization* is only one step among others in the whole capability discovery process and it may be combined with *conceptualization* the following way while attempting to satisfy a query on capabailities :

1. Try to satisfy the query Q thanks to its individualization in the assertion box (emphABox) ;
2. When no assertion satisfies Q then consider in the Box (the terminology of roles) the Most Specific and the Least Subsumer roles of Q (MSR(Q) and LSR(Q) respectively) : these may potentially contribute to the satisfaction of Q ;
3. Check again the assertion box looking for assertions which satisfy MSR(Q) and LSR(Q) ;
4. If Q is still not satisfied, then try to make it more general : go to the concept descriptions in the *TBox* (terminology of concepts) and consider the concepts in the description of which Q, LSR(Q) and MSR(Q) appear : this is what we call the *conceptualization* step ;
5. *Individualization* is then applied to every role which appear in the concept definitions

that were identified in the preceding conceptualization step. Further, when these roles are asserted as being transitive and/or symmetric, use these properties in the individualization process, if needed.

Thanks to this process, we can provide exact answers to a query on capabilities together with larger answers.

In this section, we will first illustrate unformally this process based on an example and in section 5.3 we will provide a more formal presentation of the process as we implemented it.

Therefore, in the following, based on the airline example, we successively examine :

1. the *individualization* within a single mediator, i.e. how to find out individuals who meet requested capabilities expressed thanks to roles ;
2. the *individualization* within multiple mediators,
3. and the *capability discovery*, i.e. how to combine the exploitation of the ABox, the TrBox and the TBox in the process of query satisfaction.

Individualization within a Single Mediator

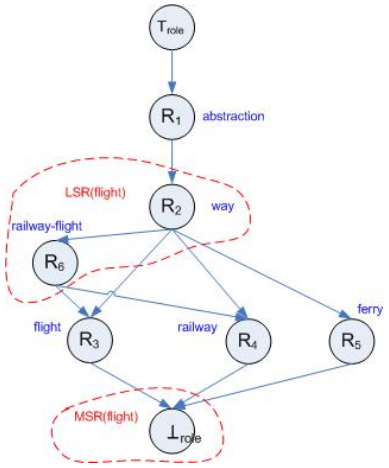
Let us recall the example on travel to detail the capability discovery process. An airline company mediator builds its knowledge system following exactly the model in figure 5.13. It owns the following ABox $\mathcal{A}_{air}$:

```
flight(Paris, Beijing), flight(Paris, NewYork),
   flight(Nancy, Lyon), flight(Paris, Lyon),
AIRLINE(CompanyA), AIRLINE(CompanyB), AIRLINE(CompanyC).
```

In addition, we introduce the capability description space $\mathcal{T}r_{air}$ in the travel knowledge base, i.e. all the *role* definitions and the role restriction definitions of $\mathcal{T}r_{air}$ (figure 5.14) together with the subsumption relationship hierarchy of the *roles* (left part of that figure).

There are 5 *primitive roles* in the $\mathcal{T}r_{air}$. They are introduced ($\dot{\sqsubseteq}$) by other primitive roles. They fix the subsumption relationship by the definitions. The role definition R_6 `railway-flight` is a defined role under the model of a role functional restriction f_6. f_6 is actually a *composite role* restriction (see definition 4.1.2 on page 74).

Step 1. (Role Individualization) We suppose that a client plans a travel from Nancy to Beijing. This travel requirement is described as the query : $Q_1 =$ `flight(Nancy, Beijing)` in the mediator system, where `flight` is a capability description. So we are looking for an assertion of role `flight(Nancy, Beijing)`. We firstly check the assertion of the role in the $\mathcal{A}_{air}$.



$$
\begin{array}{lrcl}
R_1 & \texttt{abstraction} & \dot{\sqsubseteq} & \top_{role} \\
R_2 & \texttt{way} & \dot{\sqsubseteq} & \texttt{abstraction} \\
R_3 & \texttt{flight} & \dot{\sqsubseteq} & \texttt{way} \\
R_4 & \texttt{railway} & \dot{\sqsubseteq} & \texttt{way} \\
R_5 & \texttt{ferry} & \dot{\sqsubseteq} & \texttt{way} \\
R_6 & \texttt{railway-flight} & \doteq & \texttt{railway} f_6 \texttt{flight}
\end{array}
$$

$$f_6 = \{(a,c) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists b \; \forall \texttt{flight} \; \forall \texttt{railway}.(a,b) \in \texttt{railway}^{\mathcal{I}} \wedge (b,c) \in \texttt{flight}^{\mathcal{I}}\}$$

FIG. 5.14 – The $\mathcal{T}r_{air}$ and the role subsumption hierarchy

---

$$
\begin{array}{lrcl}
\text{Ask-1 :} & \mathcal{L}_{Query1}(Q_1) & = & \texttt{flight(Nancy,Beijing)} \\
& \times & & \\
& \mathcal{A}_{air} & & \\
& \rightarrow & & \\
& \mathcal{L}_{Answer1}(Q_1) & = & \emptyset
\end{array}
$$

There are only 4 assertions of the *role* `flight` in the $\mathcal{A}_{air}$ where none of them exactly satisfies `flight(Nancy, Beijing)`. (From a relational data base system point of view, this would correspond to an empty answer to the query.) Instead of stoping the search process and returning an empty answer, the capability discovery approach we propose attempts to find a "non direct" result : the term `flight` is a capability description, so this term is searched for in the $\mathcal{T}r_{air}$ as a query for the capability discovery : $Q_2 = \texttt{flight}$.

Step 2. (Capability Discovery) In the $\mathcal{T}r_{air}$ hierarchy, we can straightforward find the Least Subsumer Roles (LSR is defined by Definition 5.1.3 on page 93) and the Most Specific Role (cf. Definition 5.1.4 on page 94) of the role `flight` (capability discovery uses the classification method to determine the $LSR(\texttt{flight})$ and $MSR(\texttt{flight})$, as explained in section 5.1.1).

Following the classification approach, the MSR of the role `flight` is the role $\bot_{role}$, i.e. there is no More Specific Role description of the role `flight` in the $\mathcal{T}r_{air}$. The LSRs of the role `flight` are the role R_2 `way` and the role R_6 `railway-flight` and they constitute some satisfactions for $Q_2 = \texttt{flight}$.

$$
\begin{array}{lrl}
\text{Ask-2}: & \mathcal{L}_{Query1}(Q_2) & = \texttt{flight} \\
& \times & \\
& \mathcal{T}r_{air} & \\
& \rightarrow & \\
& \mathcal{L}_{Answer1}(Q_2) & = \{\texttt{flight}, \texttt{way}, \texttt{railway-flight}\}
\end{array}
$$

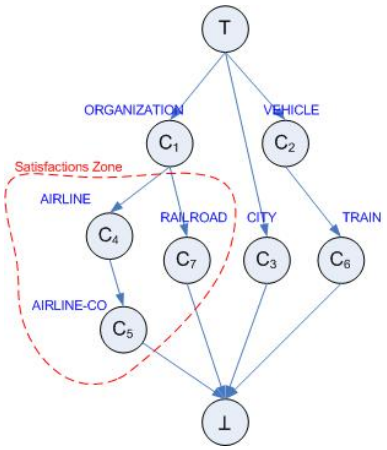
So the capability descriptions `way` and `railway-flight` are the result of Q_2 in the $\mathcal{T}r_{air}$.
Step 3. (Individualization of MSR and LSR) Let us move back the original query $Q_1 =$ `flight(Nancy, Beijing)` which is a individual discovery query. `way` and `railway-flight`, which are two capability descriptions, must be individualized to satisfy the original query. i.e., the capability descriptions must own some assertions in $\mathcal{A}_{air}$, which may finally satisfy the individual discovery Q_1. For example, if there are the assertions, `railway-flight(Nancy, Beijing)` or `way(Nancy, Beijing)`, in the $\mathcal{A}_{air}$, then the two assertions can be the satisfaction for the original query $Q_1 =$ `flight(Nancy, Beijing)`. Since these two assertions do not appear in $\mathcal{A}_{air}$, we generate two individual dsicovery queries, oen per capability : $Q_{2-1} =$ `railway-flight(Nancy, Beijing)` and $Q_{2-2} =$ `way(Nancy, Beijing)`. These queries are then addressed to $\mathcal{A}_{air}$ (Ask-2-1 and Ask-2-2) :

$$
\begin{array}{lrl}
\text{Ask-2-1}: & \mathcal{L}_{Query1}(Q_{2-1}) & = \texttt{way(Nancy, Beijing)} \\
& \times & \\
& \mathcal{A}_{air} & \\
& \rightarrow & \\
& \mathcal{L}_{Answer1}(Q_{2-1}) & = \emptyset \\
 & & \\
\text{Ask-2-2}: & \mathcal{L}_{Query1}(Q_{2-2}) & = \texttt{railway-flight(Nancy, Beijing)} \\
& \times & \\
& \mathcal{A}_{air} & \\
& \rightarrow & \\
& \mathcal{L}_{Answer1}(Q_{2-2}) & = \{\mathcal{L}_{Satisfaction}\texttt{flight(Paris, Beijing)}, \\
& & \quad \mathcal{L}_{Complement}\texttt{railway(Nancy, Paris)}\}
\end{array}
$$

The Ask-2-2 action returns a partial satisfaction `flight(Paris, Beijing)`. The answer $\mathcal{L}_{Answer1}(Q_{2-2})$ is under the *role composition* description `railway-flight`. The satisfaction is `flight(Paris, Beijing)`, and thanks to the functional restriction definition f_6 in $\mathcal{T}r_{air}$ (figure 5.14), we can calculate the *complement* which is `railway(Nancy, Paris)`. Anyway, this again constitutes a negative answer to $Q_1 =$ `flight(Nancy, Beijing)`.

Now, we know that `flight` can fully satisfy Q_2, and that `way` and `railway-flight` can partly satisfy Q_2. However there is no assertion of the three *roles* in $\mathcal{A}_{air}$ which satisfies the requirement of Q_1. Nevertheless, we identified which kinds of capability can satisfy the requirement of Q_1 and, now, we have to look for entities in $\mathcal{A}_{air}$ which hold these kinds of capability. We assume that the following concept descriptions are part of the $\mathcal{T}_{air}$

content :



C_1 ORGANIZATION $\dot{\sqsubseteq}$ $\top$

C_2 VEHICLE $\dot{\sqsubseteq}$ $\top$

C_3 CITY $\dot{\sqsubseteq}$ $\top$

C_4 AIRLINE $\doteq$ (and CITY
(all flight CITY)
(at-least 1 flight$^-$))

C_5 AIRLINE-CO $\doteq$ (and ORGANIZATION
(at-least 1 AIRLINE)
(all flight$^+$ AIRLINE)
(all flight$^-$ AIRLINE))

C_6 TRAIN $\dot{\sqsubseteq}$ VEHICLE

C_7 RAILROAD $\doteq$ (and CITY
(all railway$^+$ CITY)
(at-least 1 railway$^-$))

FIG. 5.15 – The $\mathcal{T}_{air}$

Step 4. (Conceptualization of roles) : Since $\mathcal{T}_{air}$ only accepts queries about concepts, we must conceptualize first, thanks to the universalization rule 2 (section 5.2.3), the capability descriptions which have been identified in the preceding step. For example, the capability description `flight` is rewritten as : $\forall$ `flight` A, where A is a universal concept. The universal concept corresponds to $\top$ in $\mathcal{T}$, so the concept discovery query for the capability `flight` is written as : Q_{3-1} = `all flight` $\top$. A similar way, the other two capability descriptions, `way` and `railway`, are written as : Q_{3-1} = `all way` $\top$ and Q_{3-1} = `all railway-flight` $\top$.

Thanks to the classification mechanism, we can find all the concept descriptions in the hierarchy of concepts (see the left part of figure 5.15) which hold the capabilities (They are inside the dotted-line circle in figure 5.15). This gives rise to the following Ask actions :

$$
\begin{array}{lrl}
\text{Ask-3-1 :} & \mathcal{L}_{Query1}(Q_{3-1}) & = \texttt{all flight} \top \\
& \times & \\
& \mathcal{T}_{air} & \\
& \rightarrow & \\
& \mathcal{L}_{Answer1}(Q_{3-1}) & = \{\texttt{AIRLINE-CO}\} \\
\\
\text{Ask-3-2 :} & \mathcal{L}_{Query1}(Q_{3-2}) & = \texttt{all way} \top \\
& \times & \\
& \mathcal{T}_{air} & \\
& \rightarrow & \\
& \mathcal{L}_{Answer1}(Q_{3-2}) & = \{\texttt{AIRLINE-CO}, \texttt{RAILROAD}\} \\
\\
\text{Ask-3-3 :} & \mathcal{L}_{Query1}(Q_{3-3}) & = \texttt{all railway} \top \\
& \times & \\
& \mathcal{T}_{air} & \\
& \rightarrow & \\
& \mathcal{L}_{Answer1}(Q_{3-3}) & = \{\texttt{RAILROAD}\}
\end{array}
$$

In the Ask actions, the `AIRLINE-CO` is the Most Specific Concept (MSC) of Q_{3-1}, and `RAILROAD` is the Least Subsumer Concept (LSC) of Q_{3-3}. There is no concept which directly holds `way` and which subsumes the two *roles* `flight` and `railway`. The MSCs of `way` are the concepts `AIRLINE-CO` and `RAILROAD`. Therefore, we find two satisfactions to the original query Q_1 at the level of concepts after the *Ask* actions.

Step 5. (Individualization of concepts) : A similar individualization step must then be performed : the concept descriptions `AIRLINE-CO` and `RAILROAD` must be individualized to satisfy the Q_1. Consider the two line descriptions (`all flight`$^+$ `AIRLINE`) and (`all flight`$^-$ `AIRLINE`) in the definition of the concept `AIRLINE-CO` (figure 5.15) :

- The first line description uses a capability description `flight`$^+$ which is a transitive closure of the capability `flight`. By the description, the individualization process can create some new assertion under the transitive rule. For example, if it exists two assertions `flight(Nancy, Lyon)` and `flight(Lyon, Paris)` in the $\mathcal{A}_{air}$, then they are equal to an assertion `flight(Nancy, Paris)`.
- The second line description uses a capability description `flight`$^-$ which is a symmetric closure of the capability `flight`. By the description, the individualization process can create some new assertions under the symmetric rule. For example, if it exists an assertions `flight(Paris, Lyon)` in the $\mathcal{A}_{air}$, then it exists an assertion `flight(Lyon, Paris)` too.

The two concepts, `AIRLINE-CO` and `RAILROAD`, are individualized with the condition `flight(Nancy, Beijing)` in the $\mathcal{A}_{air}$ as follows :

Ask-3-4 :
$$\begin{aligned} \mathcal{L}_{Query1}(Q_{3-4}) &= \texttt{AIRLINE-CO}\ \textit{and}\ \texttt{flight(Nancy, Beijing)} \\ &\times \\ &\mathcal{A}_{air} \\ &\rightarrow \\ \mathcal{L}_{Answer1}(Q_{3-4}) &= \{\texttt{flight(Paris, Beijing), flight(Nancy, Lyon),} \\ &\quad \texttt{flight(Paris, Lyon),} \\ &\quad \texttt{AIRLINE(CompanyA), AIRLINE(CompanyB),} \\ &\quad \texttt{AIRLINE(CompanyC)}\} \end{aligned}$$

Ask-3-5 :
$$\begin{aligned} \mathcal{L}_{Query1}(Q_{3-5}) &= \texttt{RAILROAD}\ \textit{and}\ \texttt{flight(Nancy, Beijing)} \\ &\times \\ &\mathcal{A}_{air} \\ &\rightarrow \\ \mathcal{L}_{Answer1}(Q_{3-5}) &= \emptyset \end{aligned}$$

Applying the transitive and symmetric rules, the three assertions of `flight`, which are `flight(Paris, Beijing)`, `flight(Nancy, Lyon)`, and `flight(Paris, Lyon)`, implies `flight(Nancy, Beijing)`. An individual of `AIRLINE-CO` may satisfy the requirement of Q_1. Then the individuals $\mathcal{L}_{Answer1}(Q_{3-4})$ who satisfy Q_{3-4} are :

$$\begin{aligned} \mathcal{L}_{Answer1}(Q_{3-4}) = \{&\texttt{and}\ \ \texttt{AIRLINE(CompanyA)} \\ &\texttt{AIRLINE(CompanyB)} \\ &\texttt{AIRLINE(CompanyC)} \\ &\texttt{flight(Nancy, Lyon)} \\ &\texttt{flight(Paris, Lyon)} \\ &\texttt{flight(Paris, Beijing)}\} \end{aligned}$$

$\mathcal{L}_{Answer1}(Q_{3-4})$ fully satisfies the requirement of the original query Q_1. The capability requirement `flight(Nancy, Beijing)` is satisfied thanks to the transitive and symmetric closure of the capability `flight` (figure 5.16 sketches all the steps which we have mentioned for this travel example).

We believe that the $\mathcal{L}_{Answer1}(Q_{3-4})$ is the best answer for the original query in the airline travel mediator system. However, in a *"satisfaction first" strategy*, this answer can be acceptable since $\mathcal{L}_{Answer1}(Q_{3-4})$ is a full satisfaction. But if the mediator system is constrained by an other strategy, which for example limits the number of individuals to *"at most 2"*, then $\mathcal{L}_{Answer1}(Q_{3-4})$ is not retained as a full satisfaction by the system. In a federation of mediators, the system will cooperate with other mediators to find a satisfaction which is conform to any imposed strategy.

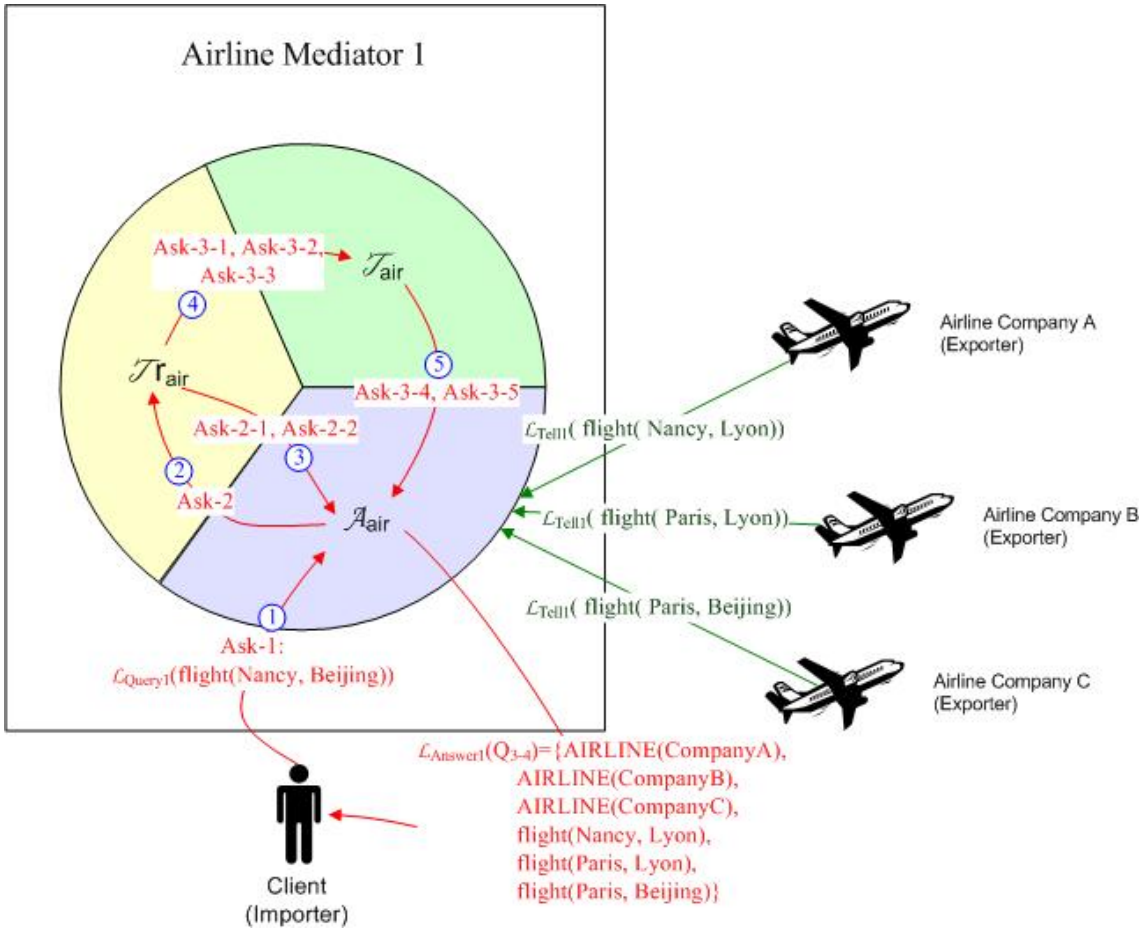


FIG. 5.16 – The steps of capability discovery in the airline mediator

Individualization within Multiple Mediators

We have to send the missing *complement* to an other mediator. The Ask-2-2 action has determined a complement `railway(Nancy, Paris)` and the corresponding answer is expressed as :

$$\mathcal{L}_{Answer1}(Q_{2-2}) = \{\mathcal{L}_{Satisfaction1}(\texttt{flight(Paris, Beijing)}), \mathcal{L}_{Complement}(\texttt{railway(Nancy, Paris)})\}$$

The "airline mediator 1" (figure 5.16) uses a family of languages $\mathcal{L}_1$. This family of languages consists of the query language $\mathcal{L}_{Query1}$, the reply language $\mathcal{L}_{Answer1}$, and the update language $\mathcal{L}_{Tell1}$. However, we limit ourselves to the three description languages ($\mathcal{L}_{Satisfaction}$, $\mathcal{L}_{Complement}$, $\mathcal{L}_{Difference}$) which are actually used in capability description. The airline travel mediator owns a knowledge base in $\mathcal{ALN}_{r+}$ which consists of the TBox $\mathcal{T}_{air}$, the TrBox $\mathcal{T}r_{air}$, and the ABox $\mathcal{A}_{air}$.

Now, suppose we have a railway travel mediator which uses a different set of languages $\mathcal{L}_2$ to express queries and answers (For example, the railway travel mediator builds and manages its knowledge base in F-logic). We call it "railway mediator 2" in figure 5.17.

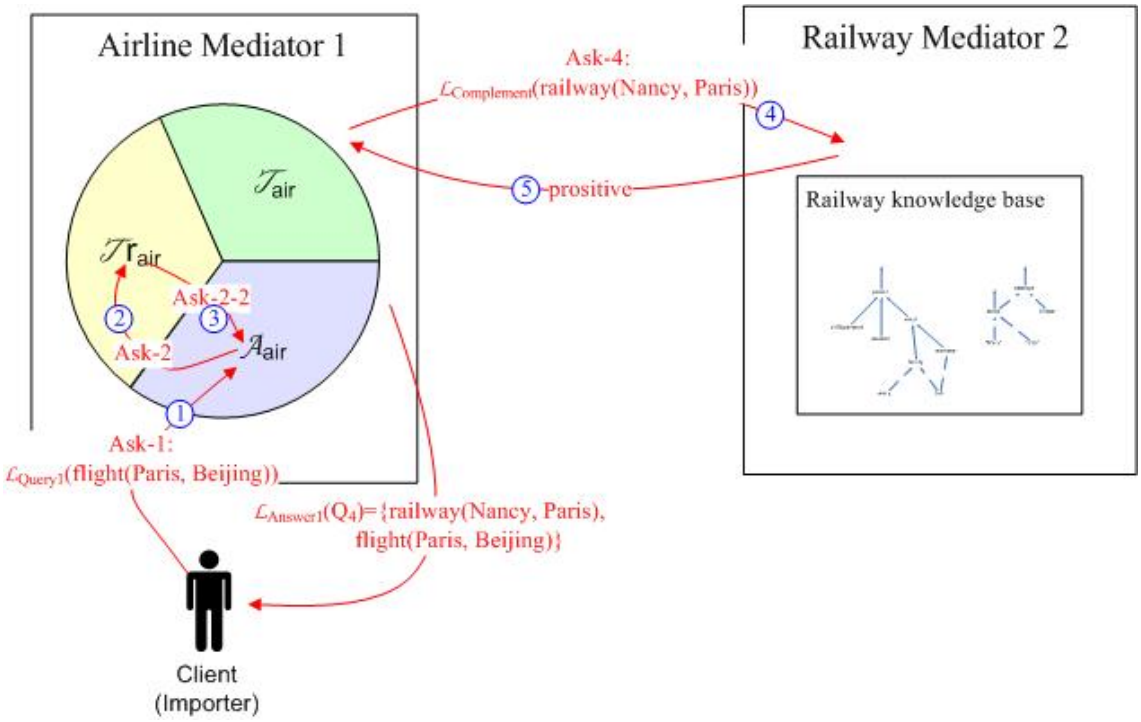


FIG. 5.17 – The steps of capability discovery in two mediators

The railway mediator 2 firstly checks the vocabularies in the capability discovery description $\mathcal{L}_{Complement}$(`railway(Nancy, Paris)`). If the term which describes the capability is not in the knowledge base of the mediator 2, or the capability is described in a foreign language, then it has to be translated using the method which was introduced in the previous section. For example, if we suppose that the railway mediator 2 is in French, then the capability description will be translated into `train(Nancy, Paris)`.

In addition, since the railway mediator 2 is in F-logic, the query of capability discovery is rewritten in F-logic :
$? - \ X : train \wedge X[$ départ$\rightarrow\rightarrow D[nom \rightarrow$ "$Nancy$"]; arrivée$\rightarrow\rightarrow A[nom \rightarrow$ "$Paris$"]]

If the query is satisfied in the knowledge base, then the mediator 2 sends a positive answer to mediator 1. Therefore, the mediator 1 can deliver a new full satisfaction to the client. This answer is expressed in the original $\mathcal{L}_{Answer1}(Q_4)$ as :

$\mathcal{L}_{Answer1}(Q_4)$ = { `railway` f_6 `flight` (`railway(Nancy, Paris)` `flight(Paris, Beijing)`)}

The steps of this heterogeneous capability discovery are shown in figure 5.17.

Individualization and Conceptualization : Concluding Remarks

This section shows the processes of individualization and conceptualization thanks to a driving example. Thanks to these processes, we can find the satisfaction for a given query at the three components of a knowledge base : the terminology of concepts (TBox), the terminology of roles (TrBox) and the assertions (ABox). Further, it may find answers which are described in heterogeneous knowledge bases, thanks to a simple structure of capability description.

Intuitively, individualization may deliver a complete answer. Thanks to the previous approaches, we locate some *concept* descriptions and roles *role* descriptions in the knowledge base, which possibly satisfy the query requirements. All the assertions of *roles* and *concepts* are candidates for a composite answer. This process is more formally expressed in the next section.

5.3 The Individualization Process and its Implementation

The *satisfaction* description consists of two kinds of atomic concept structures : primitive concept and capability description structure. The UML interaction diagram in figure 5.18 sketches the individualization process thanks to two reference frames : `IndividualizePrimitiveConcept` and `IndividualizeCapabilityDescription`.

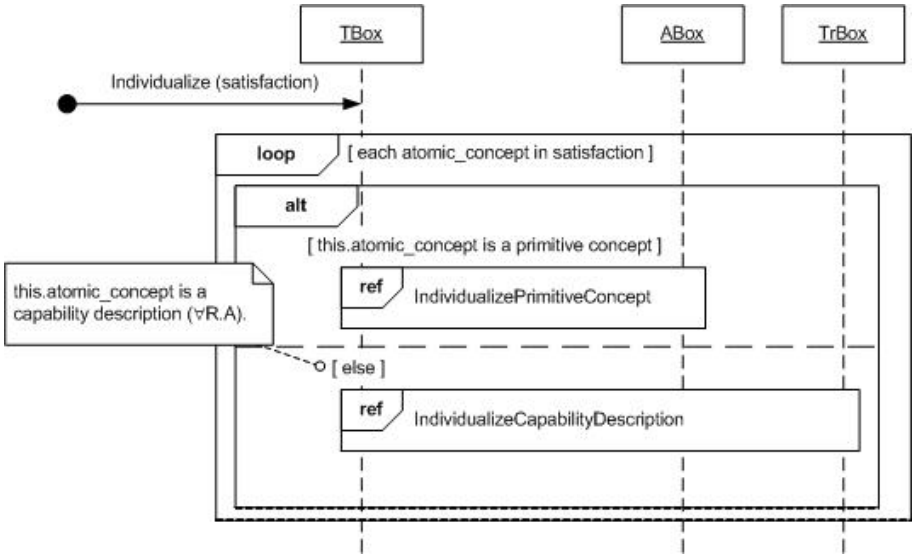


FIG. 5.18 – The interaction diagram of individualization

The individualization process always applies to the *satisfaction* description under its normal form. It exists two kinds of atomic structures in the $\mathcal{ALN}_{r+}$ normal form concept

description, which are the primitive concept A and the number restriction $\geq m\ n\ R\ A$. In the heterogeneous discovery approach, the number restriction is translated into the capability description structure "$\forall R.A$". In the following paragraphs, we successively detail :

1. the individualization process of primitive concept descriptions,
2. the individualization process of capability description structure,
3. the transitive role discovery process.

5.3.1 Primitive concept individualization

We sketch first the individualization process of primitive concept descriptions as shown in figure 5.19.

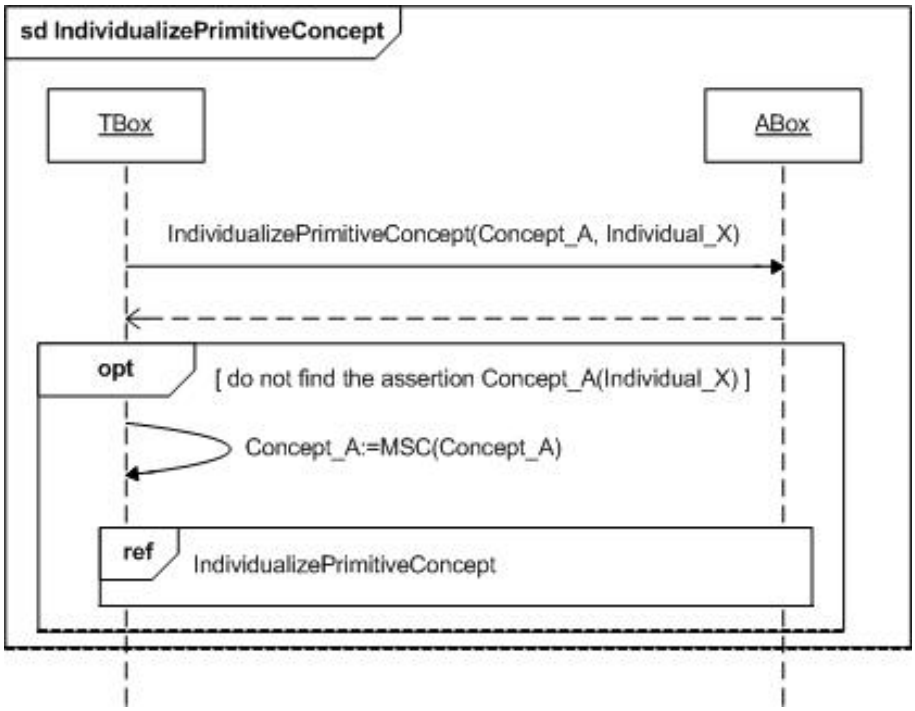


FIG. 5.19 – The sub-diagram of IndividualizePrimitiveConcept

1. The process `IndividualizePrimitiveConcept(Concept_A, Individual_X)` has two input parameters : `Concept_A` and `Individual_X`. `Concept_A` is the identification of the primitive concept which we are looking for. `Individual_X` identifies an individual which we want it is under the `Concept_A`. Thanks to the two parameters, we can locate the assertion `Concept_A(Individual_X)` in $\mathcal{A}$. For example, if `Concept_A` is `CITY-TRAIN` and `Individual_X` is `Nancy`, then we search for the assertion description `CITY-TRAIN(Nancy)`.
2. If the assertion `Concept_A(Individual_X)` does not exist in the $\mathcal{A}$, then we have to find the Most Specific Concept (MSC) of the `Concept_A`, and recall the

`IndividualizePrimitiveConcept` process with `MSC(Concept_A)`
i.e. `IndividualizePrimitiveConcept(MSC(Concept_A), Individual_X)`.
For example, the original process `IndividualizePrimitiveConcept(CITY, Nancy)` does not exist in the $\mathcal{A}$. The `MSC(CITY)` being `CITY-TRAIN`, the individualization process will be `IndividualizePrimitiveConcept(CITY-TRAIN, Nancy)`.

3. If `MSC(Concept_A)` is $\perp$, then we stop the individualization process.

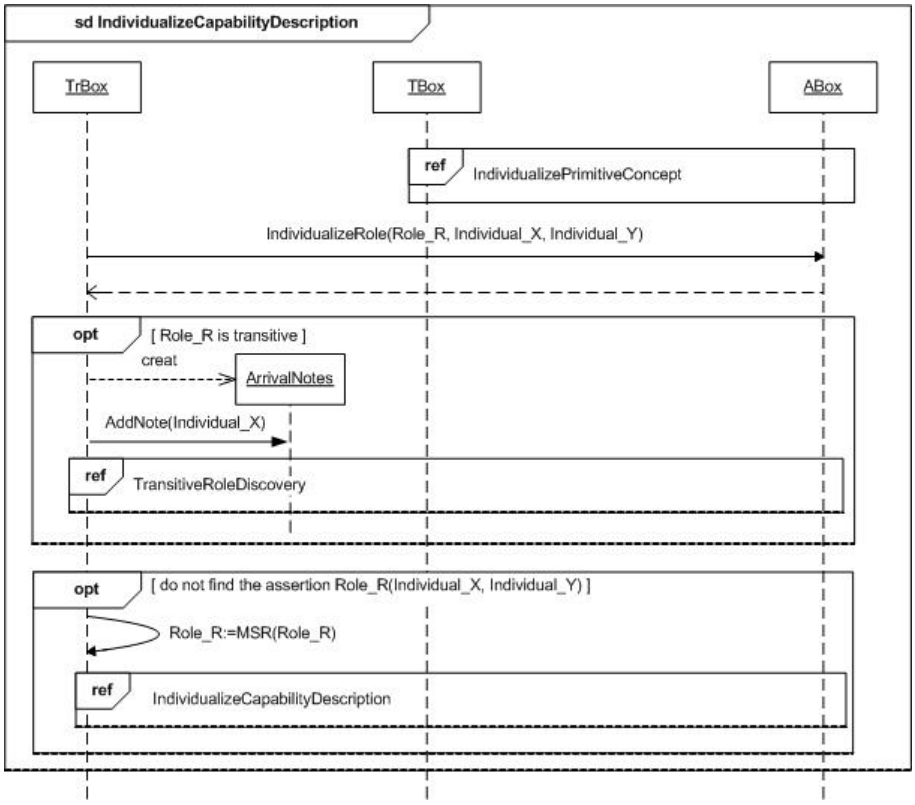


FIG. 5.20 – The sub-diagram of IndividualizeCapabilityDescription

5.3.2 Capability description Individualization

Secondly, we sketch the individualization process of capability description structure ($\forall R.A$) as described in figure 5.20.

- The capability description relates two term descriptions : role description R and concept description A. The concept description A is a primitive concept, so we ahead recall the *IndividualizePrimitiveConcept* process for A.
- The process `IndividualizeCapabilityDescription()` is like the primitive concept individualization process, but it has three input parameters : `Role_R`, `Individual_X`, and `Individual_Y`. `Role_R` is the identification of the primitive role. An assertion of role relates two individuals ($R(X, Y)$), so there are two individual parameters : `Individual_X` and `Individual_Y`. So, ignoring the transitive role process frame in

the middle of figure 5.20,
`IndividualizeCapabilityDescription()`
is similar with
`IndividualizePrimitiveConcept()`.

– When the original role cannot be satisfied, we recall *IndividualizeCapabilityDescription()* with the Most Specific Role (MSR).

5.3.3 Transitive role discovery process

Finally, we focus on the transitive role discovery process frame in figure 5.20. The `Role_R` is transitive : that means that several assertions can be reorganized into a new assertion. For example, if there are `has-flight(Paris, Beijing)` and `has-flight(Beijing, Wuhan)`, then it exists `has-flight(Paris, Wuhan)`. Further, if a composition relationship exists between two different roles ($R \circ S$), then they are transitive at the individualization level. For example, if there are `has-flight(Paris, Beijing)` and `has-train(Nancy, Paris)`, then it exists `has-train`$\circ$`has-flight(Nancy, Beijing)`.

Before starting the transitive role discovery process, an individual set of objects `ArrivalNotes` is created, and the `Individual_X` is added into the `ArrivalNotes`. The `ArrivalNotes` will hold all individuals which can be reached by the role `Role_R` "starting from" from the individual `Individual_X`. The transitive role discovery process is sketched in figure 5.21.

There are two loop frames in the process. *(1)* The first loop, with the guard `[i<ArrivalNotes[].size]`, looks for all individuals which can be attained for the current set of individuals in `ArrivalNotes[]`. `IndividualizeRole(Role_R, ArrivalNotes[i])` finds all the assertions which are under the `Role_R` and starts from individual `ArrivalNotes[i]` (i.e. `Role_R(ArrivalNotes[i], X), where X is any individuals`). All the assertions are returned in an output parameter `Arrival[]`.

(2) The second loop with the guard `j<Arrival[].size`, checks whether the individuals in `Arrival[]` already exist in the `ArrivalNotes[]`. If they do not, the individuals are added in the `ArrivalNotes[]`.

After the two loop processes, we check whether `Individual_Y` exists in `ArrivalNotes[]`. If it is true then the transitive role `Role_R` has a "path" from `Individual_X` to `Individual_Y` in the $\mathcal{A}$, else if the `ArrivalNotes[].size` is not changed after one times of `TransitiveRoleDisocvery` process, that mean no more individual can be arrival. In else case, we will recall the `TransitiveRoleDisocvery` process.

For the example `has-way(Nancy, Wuhan)`, by the definition, we know that the roles `has-flight` and `has-train` are transitive in the concept definitions, and that the *composite role* relationship holds between `has-flight` and `has-train` (`has-flight`$\circ$`has-train`). So by the previous UML inter-action diagram, we can find a transitive role discovery which relates the three role assertions : `has-train(Nancy, Paris)`, `has-flight(Paris, Beijing)`, and `has-flight(Beijing, Wuhan)`. Finally, we find the *assertions* of *concepts* which holds the *role* in the composite answer. This answer relates to four *concepts* : `CITY-TRAIN(Nancy)`, `CITY-AIR-TRAIN(Paris)`, `CITY-AIRPORT(Beijing)`, and `CITY-AIRPORT(Wuhan)`. A final answer is then :

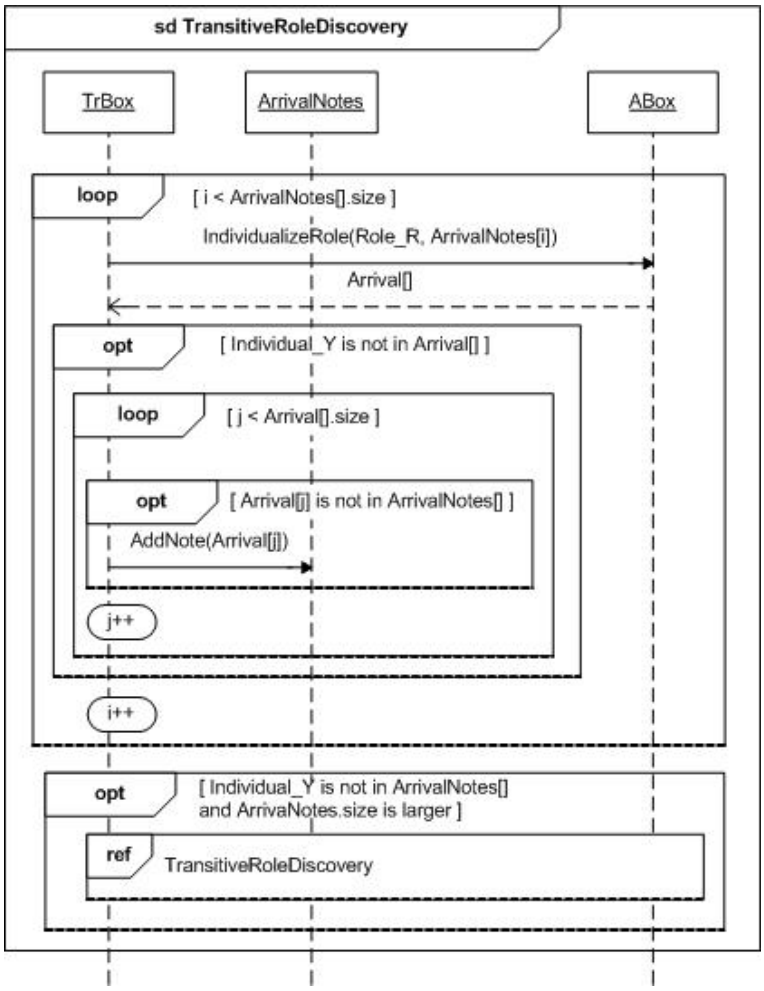


FIG. 5.21 – The sub-diagram of role transitive

$\mathcal{L}_{Answer}(Q)$ = {and CITY-TRAIN(Nancy)
has-train(Nancy, Paris)
CITY-AIR-TRAIN(Paris)
has-flight(Paris, Beijing)
CITY-AIRPORT(Beijing)
has-flight(Beijing, Wuhan)
CITY-AIRPORT(Wuhan)}

This answer is a full satisfaction of the query Q=has-way(Nancy, Wuhan).

5.4 Conclusion

This chapter detailed the approach we propose for *capability management* and *capability discovery*. The $\mathcal{ALN}_{r+}$ knowledge base consists of three parts : TBox, T$_r$Box, and ABox.

We opted for the classification approach for the maintenance of the hierarchies (section 5.1). The classification approach uses a normalization comparison algorithm (NC algorithm) for testing the subsumption relationship. The result of the NC algorithm is applied in the composite answer calculation. The NC algorithm is sound and complete in $\mathcal{ALN}_{r+}$, but it is not in all DL languages. Other algorithms (like tableau-based algorithms [6]) may be used for these calculations too, and it work in most DL languages.

The results of the comparison algorithm are put in a *satisfaction table* which is analyzed for determining a composite answer. We implemented some composite answer strategies in the *satisfaction table*, but it is not enough for some capability discovery without composite answer (like the disjoint concept discovery). We need a more general capability discovery strategy description method for them.

These approaches are expanded to deal with heterogeneous environments. We introduced the capability composition of individuals in section 5.2.4. We mentioned three knowledge representation languages in this experimental work. The capability description structure ($\forall R.A$) is manually translated in the three knowledge representation languages. It may need some automatic/semi-automatic approaches to implement the capability description structure translation.

We detail these approaches by some formula inferences and the travel example in this chapter. However, the capability discovery approach can be easily applied in many knowledge systems, especially in heterogeneous environments. As an illustration, in the appendix (see section A), we provide an additional example of this approach based on the interoperability domain knowledge.

We developed a prototype system to validate our proposals and the next chapter details this prototype system implementation.

Chapitre 6

Mediation Federation

We implemented a mediator federation prototype system to illustrate our proposals. The prototype consists of two parts : *(i)* the capability application approaches in $\mathcal{ALN}_{r+}$ Knowledge Base, and *(ii)* the mediator server composition in heterogeneous environment.

In the first part, all the facilities, which were mentioned in the previous chapter and that are required by a capability management application, has been developed in *Java*. These include :

- a service for testing the subsumption relationship,
- a service for the determination of the *complement* and the *satisfaction* concepts,
- and a service for the calculation of the *composite answers*, when required.

In the second part, we experimented composite heterogeneous mediation services in an environment as drawn in figure 6.1. The three federated mediators show a heterogeneous and distributed environment where :

- Mediator 1 supports the increased DLs language $\mathcal{ALN}_{r+}$, which loads all the programs of capability application in the first part ;
- Mediator 2 supports F-Logic, and its communication language format is KIF [41] ;
- Mediator 3 supports Conceptual Graphs (CGs), and its communication language is CGIF [64].

The federated mediator architecture conforms to the Service Oriented Architecture (SOA) [5].

In this chapter, we partly detail the implementation of the two parts. In section 6.1, we focus on the *inside* of the mediator server in $\mathcal{ALN}_{r+}$. The mediator server consists of four parts (see figure 6.1) :

1. a local repository,
2. a reasoning processor,
3. a syntax translator,
4. and a lexical ontological dictionary.

In section 6.2, we focus on the *outside* of the mediator servers, as an heterogeneous environment, and we will detail the Web service technologies that served for the composition between the heterogeneous mediator servers.

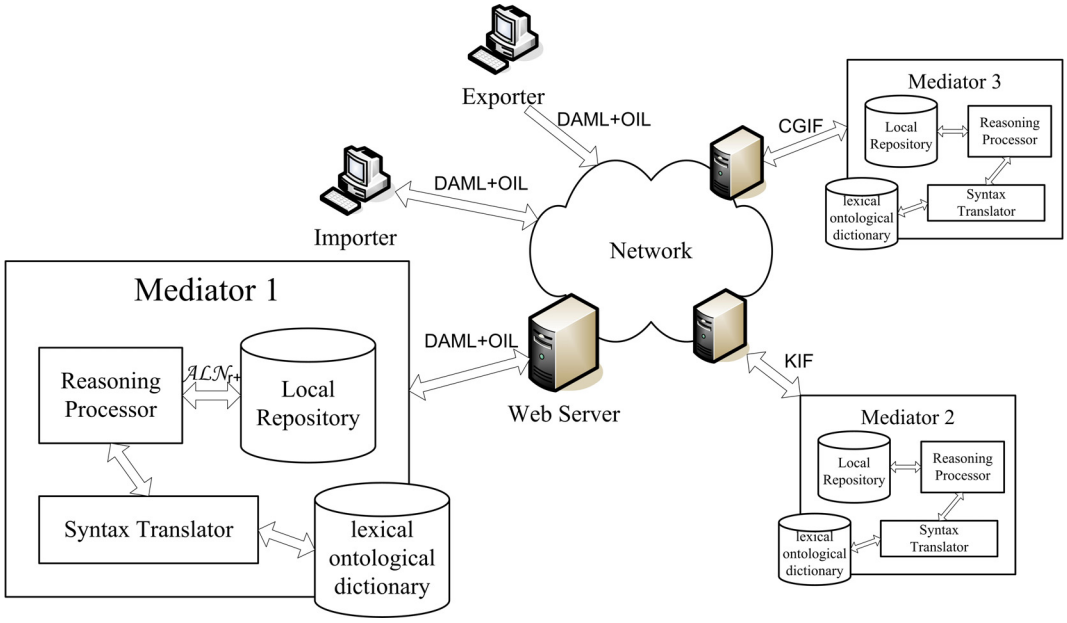


FIG. 6.1 – The Sketching of Mediation Architecture

6.1 The Mediator Server

In the Mediator Architecture in figure 6.1, the *Exporter* and *Importer* are the two kinds of clients of the *Mediator Federation*. The *Exporter* binds with the *Tell* action, which is the capability provider. The *Importer* binds with the *Ask* action, which is the capability discoverer. In this work, *exporter* and *importer* are two abstract roles. They can be any software or nature entity, which can send a capability description or a capability requirement in $\mathcal{ALN}_{r+}$ using the *Tell* and the *Ask* actions.

For experimenting mediator federation, we considered three mediators (figure 6.1) which support three different Knowledge Representation Languages : DL, F-Logic, and CGs which have been experimented in this work to illustrate heterogeneity. The mediator services are described in WSDL [56], and the composition between the mediators are defined in OWL-S [37].

We totally implemented the mediator server in $\mathcal{ALN}_{r+}$, i.e. the mediator server owns a repository in $\mathcal{ALN}_{r+}$. A reasoning processor, a syntax translator and a lexical ontological dictionary are also available. The mediator server accepts the *Description Logical* language $\mathcal{ALN}_{r+}$ to reason and represent knowledge and this language is used to represent the individuals, the entities, and the capability of entities, in the local repository. From an implementation perspective, the *Java* objects (see figure 5.6) of $\mathcal{ALN}_{r+}$ are used in all the implemented algorithms.

Furthermore, the mediator services (*Tell* and *Ask*) are implemented as Java Servlet CGI programs. These service interfaces are described in WSDL, that enables the other mediators to access these services (see the next section for further details on the Web Service technology we used). The approach for capability management and capability

discovery, we mentioned in the previous chapters, are mainly implemented in a module of the *local repository* and the *reasoning processor*. The syntax translator simply works for one sentence of *capability description*. It needs the lexical ontological dictionary to support mapping of terms (we opted for WordNet [65] as a dictionary of terms). The two following sections successively detail the *local repository* and the *reasoning processor* implementation.

6.1.1 The Local Repository

The local repository implements the knowledge base, which we mentioned the conceptual model in section 4.2, on page 77. The knowledge base uses the knowledge representation language $\mathcal{ALN}_{r+}$ which defines the syntax of concept and role descriptions.

We designed two service interfaces of this repository : the *Tell* and the *Ask*, which are implemented by `Repository.Tell()` and `Repository.Ask()`.

The `Tell()` operation is the only action which introduces new knowledge into the repository. Before introducing any new concept description and any new role description into the TBox and TrBox, the descriptions are normalized, according to the rules and the normalization process we have introduced in section 5.1.2 (page 95). Therefore, all the *concept* descriptions and the *roles* descriptions are stored in the repository under their normal forms (see figure 5.6 on page 98).

On the other hand, the `Ask()` operation is the only way to access and query the knowledge in the repository. The `Ask()` operation result is a composite answer, and gives the *case of composite answer* of this answer. These results will be used by a reasoning processor. The whole repository architecture is sketched in the UML class diagram in figure 6.2 that is the basis for the implemented algorithms.

We introduce hereafter the main algorithms we have implemented. In section 5.1 (page 88), we introduced a normalization comparison algorithm, which is the kernel of the subsumption test and the *complement* concept calculation. We also introduced the rules of three normalization sub-processes : atomization, normalization, and simplification (see figure 5.4 on page 97). These functions are implemented on the date structure in the repository by the algorithm which is presented in figure 6.3.

As said earlier (page 98), the normalization process reduces the "before normalization" class diagram in figure 5.5 into the "after normalization" class diagram in figure 6.5. The obtained normal forms can further be used as inputs in the subsumption relationship test algorithm which is exposed in figure 6.6.

One should notice that we did not implement the approach in other logic languages since we consider that the approach may be implemented a similar way in the other logic languages. The subsumption relationship is the kernel inference, which is a common concept in all knowledge representation systems : for example, it corresponds to the subclass concept in F-Logic, and to the sub-graph concept in CGs.

The results of the repository inferences are further used in the reasoning processor.

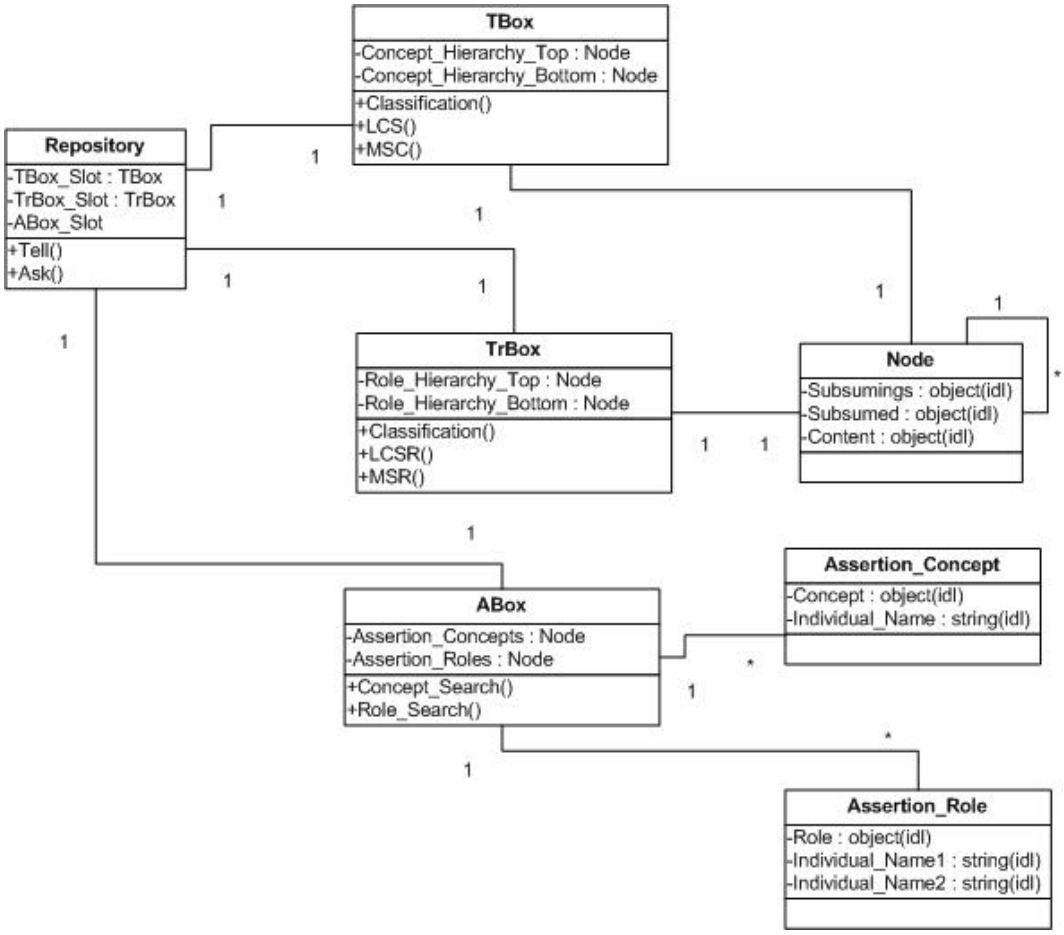


FIG. 6.2 – The UML Class Diagram of the Local Repository

```
//Normalize process
//It includes 3 steps : atomization, normalization, and simplification

void Normalize (Concept Cp) {
//Cp is a pointer at the data structure of the concept hierarchy

    //The first step is the concept hierarchy atomization
    Cp=Atomize(Cp) ;

    //The second step is the normalization of all the sub-objects
    //Scan all the sub-objects of Cp by Normalize()
    for ( Every concept in Cp.Subsumings)
        Normalize (Cp.Subsumings.next()) ;

    //The third step is the concept hierarchy simplification
    Cp=Simplify(Cp) ;
}
```

FIG. 6.3 – Atomization, Normalization and Simplification in the $\mathcal{ALN}_{r+}$ Repository

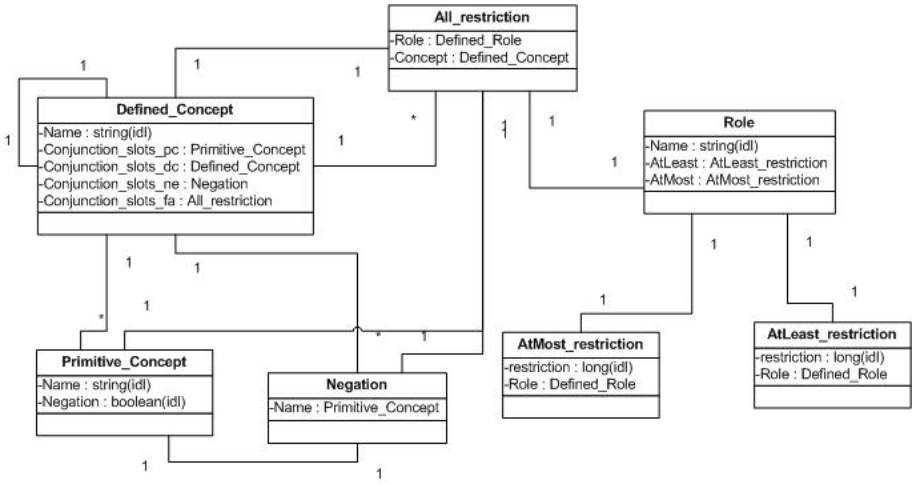


FIG. 6.4 – Description Form in $\mathcal{ALN}_{r+}$, before Normalization

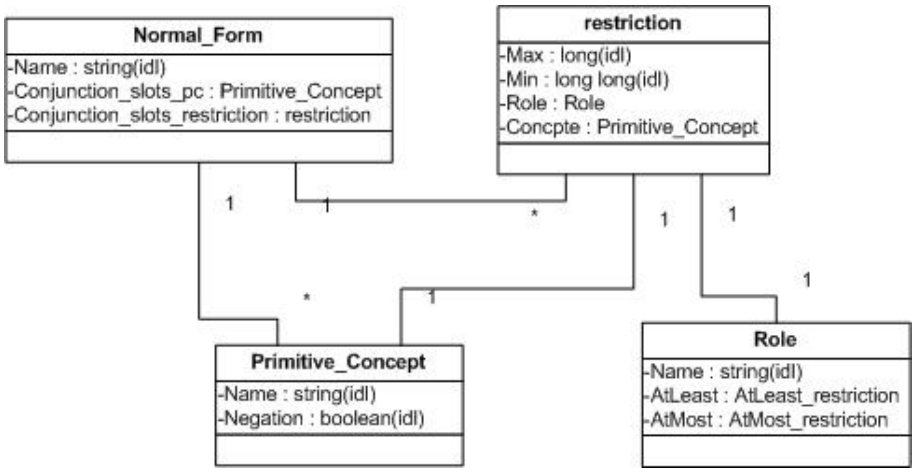


FIG. 6.5 – Description Form in $\mathcal{ALN}_{r+}$, after Normalization

```
//Test whether C subsumes D
//Cp is a pointer at data structure of satisfaction concept
//Dp is a pointer at data structure of query concept
//Detail will be used to determine the complement
boolean Subsume (Concept Cp, Concept Dp, boolean[] Detail){
    boolean SbC = True ; // SbCi note D ⊑ C
    for ( Every concept in Cp.Subsumings){
        boolean SbCi = False ; // SbCi note D ⊑ Ci
        for ( Every concept in Dp.Subsumings){
            switch (Cp.RulesID){
                //here is pass over some tests by these rules,
                //the result is the boolean value True
                SbCi = (SbCi ∨ True) ;// SbCi denotes D ⊑ Ci
            }
        }
        Detail[Cp.index] = SbCi ;//contains every Ci subsumption case
        SbC = SbC and SbCi ;// when all SbCi are true, SbC is true
        Cp.index = Cp.index + 1 ;//next atomic concept
    }
    return SbC ;// return value de SbCi, it is case of D ⊑ C.
}
```

FIG. 6.6 – The comparison between the $\mathcal{ALN}_{r+}$ normal form concepts

6.1.2 The Reasoning Processor

The reasoning processor is the task context manager and the interface of the service. The mediator federation is a distributed query/answer system which needs that the reasoning processor manages the query/answer tasks which are distributed to other mediator servers. The reasoning processor also includes some services (implemented as CGI programs in Java Servlets) which can be accessed by the *importers*, the *exporters* and by other mediator servers as well. According to the conceptual model of the mediator service (see figure 3.1 on page 60), we implemented the two main operations in the *reasoning processor* : `export()` and `import()`. We successively introduce the details of these two operations hereafter.

Regarding the *export* operation :

The UML sequence in figure 6.7 sketches the following process :

1. The *exporter* sends its capability description encoded in $\mathcal{ALN}_{r+}$ to the *Reasoning Processor* : The *exporter* accesses the Mediator server (thanks to a *Web Server CGI* program, that conforms to the standard of the Web client/server architecture). The mediator server accepts the $\mathcal{ALN}_{r+}$ *concepts* written in *DAML+OIL*, an ontology language standard in *XML* [52].
   The $\mathcal{ALN}_{r+}$ *concept* descriptions, under their *DAML+OIL* encoding, are transmitted to the mediator server's task manager (a CGI program) as a parameter.
2. The *CGI* program is in charge of all the export/import tasks. It uses the *Syntax Translator* to translate the *DAML+OIL* text to $\mathcal{ALN}_{r+}$ *concept* objects, as figured by the *daml2alnr*$^+$*()* (DAML+OIL to $\mathcal{ALN}_{r+}$) method on figure 6.7.
3. Then the *Reasoning Processor* creates a *Tell* task with the parameter of the *concept* objects. The *Reasoning Processor* calls the *Classification* inference mehod, which is provided by the local repository, to add the new *concept* into the local knowledge base.

Regarding the *import* operation :

In the mediator federation environment, the *Import* process is the most important and the most complex one. The query may require accessing multiple mediator servers which themselves may support different knowledge representation languages. The mediator servers results are finally reorganized into a composite answer which is the final answer for the query in the mediator federation. From the mediator perspective, the query task may receive requests from two different roles : *(i)* the importers and *(ii)* the other mediator servers which are in the federation. The two kinds of requests are processed the same way by the mediator :

1. The reasoning processor creates a *satisfaction table* for each *Ask* tasks, i.e. for each query arriving at a mediator server. The *satisfaction table* will hold the information about the satisfaction of the query, as explained in chapter 5.

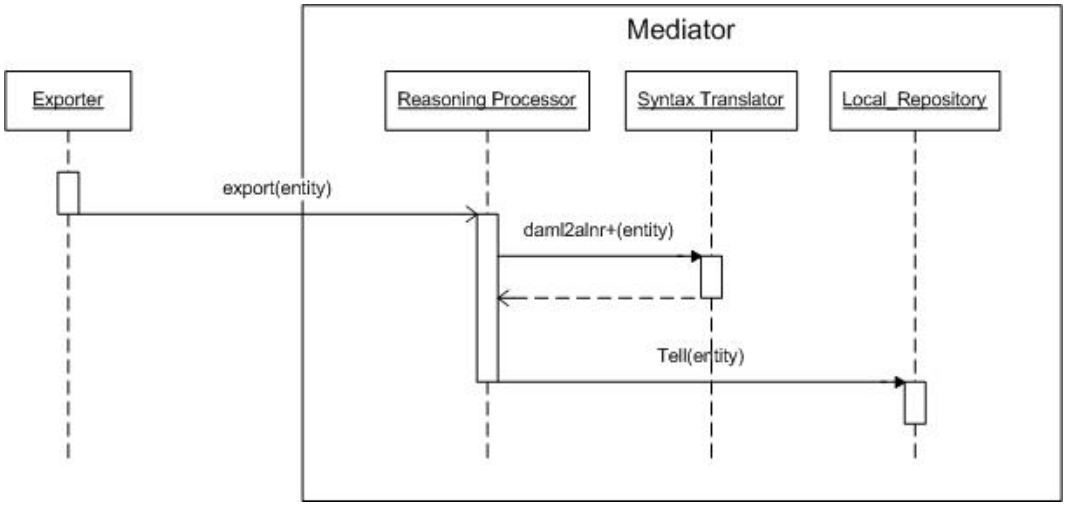


FIG. 6.7 – The UML Sequence of the Export Process

2. At the same time, while creating the *satisfaction table*, the concept descriptions in the query are normalized in the local repository. In this step, the mediator server can accept heterogeneous query concept descriptions thanks to the *Syntax Translator*.

The translator is working at the syntax level of the *capability description*. Naturally, the translator also includes a lexical translator (as mentioned in section 5.2 on page 100). In fact, syntax translation as well as lexical translation are taken in charge by the *Syntax Translator* in the prototype system.

Syntax Translator and Lexical Translator

The *syntax translator* rewrites query descriptions in different query language formats. $\forall R.\top$, a simple capability discovery model for the communication, is defined between the mediators, R being a requested capability and $\top$ meaning *"any entity which satisfies R"* (see section 5.2.3, on page 109, for further details). The translation rules of this simple sentence between multiple query languages can be manually defined (as shown in section 5.2.3, on page 5.2.3) : actually, we considered three knowledge representation languages : DLs, F-Logic, and CGs. We also accept multiple interchange standard formats like XML, KIF, etc., for the knowledge representation languages.

The process of term translation need to be supported by a lexical dictionary. When an unknown term, which is defined in the lexical dictionary, is in the query concept description, we try to find a similar term in the local repository. In the prototype system, we opted for WordNet, which is a hierarchical lexical data base in English. The *syntax translator* of a mediator calculates the similarity of the unknown term with all the capability description terms in the *local repository* and it retains the term with the highest similarity measure.

Let us move back to the sequence of the import process which is sketched in figure 6.8. The *Reasoning Processor* and the *Syntax Translator* makes sure that the query concept can be accepted by the *Local Repository*. We can put the query concept description into

the *Local Repository* to start the discovery approaches which are detailed in the chapter 5.

The prototype follows the *satisfaction first* strategy which we mentioned in the chapter 3.2.2. When the *Composite Answer Situation Case* is under the 4 (Partial Satisfaction) and 5 (Failure) the (see section 3.2.2), the *Reasoning Processor* will send the *complement concept* as a query to the other mediation service partners. The prototype system includes an abstract object *Mediator Federation* which takes charge all the mediator server discovery and management jobs. The service discovery and management is real huge related works in the network management domain. In the prototype, we implement the *Mediator Federation* in Web Services technic, that will be introduce in the next section. The *Mediator Federation* will send the *complement concept* to all the mediator servers in the federation. The *Import Process* will replay in all the mediator server partners in the federation, which get the queries.

At the end of the *Import Process*, all the results which are rendered by the other mediator servers in the federation, are put in the *satisfaction table*. The *Reasoning Processor* analyzes the *satisfaction table* to give the final answer for the query.

Reasoning Processor and Translator : Concluding Remarks

In this section, the two UML sequence graphs showed the detail of two processes : the *import* and the *export*. The *Syntax Translator* translates the capability description which is in the query, and the *Reasoning Processor* manages the import task. From an implementation point of view, thanks to an abstract object[9] *Mediator Federation*, the federation management can be implemented in multiple techniques. This prototype system implements the *Mediator Federation* abstract object in Web Services techniques (which were introduced in section 2.3.1) [5, 106]. Details about the implementation of the *Mediator Federation* abstract object are given in the coming section.

6.2 Mediator Federation

The mediator federation is an abstract architecture, which can be implemented in many different distributed technologies. In addition, some work results are represented in *Protégé* [43], a popular visual ontology editor tools.

The ontologies in *Protégé* are frame-based. *Protégé* support a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, and a set of individuals those classes. The concept hierarchy is a tree structure, where one concept has only one subsumer concept. That is different from the concept hierarchy in $\mathcal{ALN}_{r+}$, and the capability description hierarchy, as a role hierarchy, does not exist. The concept hierarchy and the individuals in $\mathcal{ALN}_{r+}$ can be imported in *Protégé* (see figure 6.9 as an example of the `City` concept description).

This section elaborate more on some particular aspects of the implementation technology :

[9] Abstract object is to be understood here in its meaning in Object-Oriented Programming.

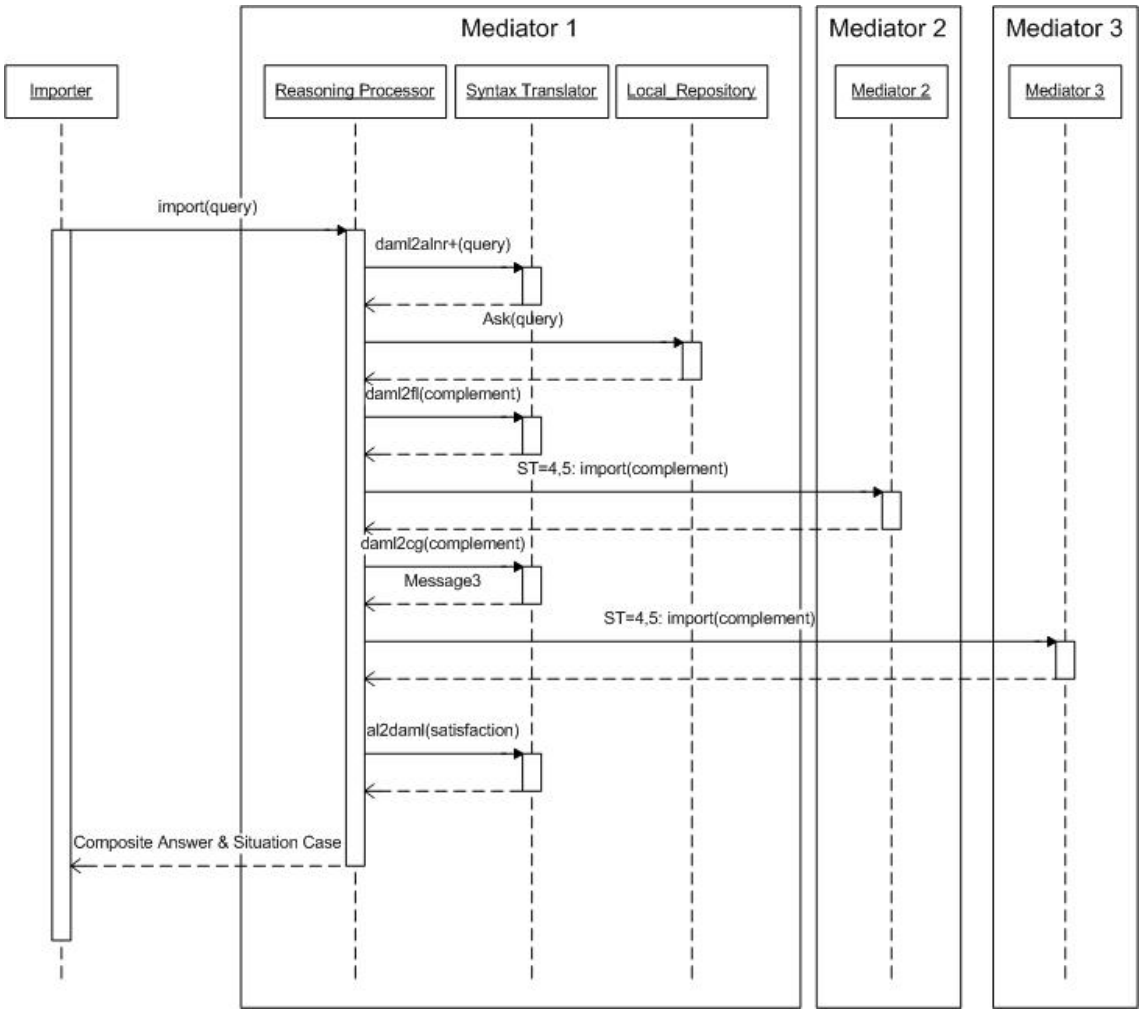


FIG. 6.8 – The UML Sequence of Import Process

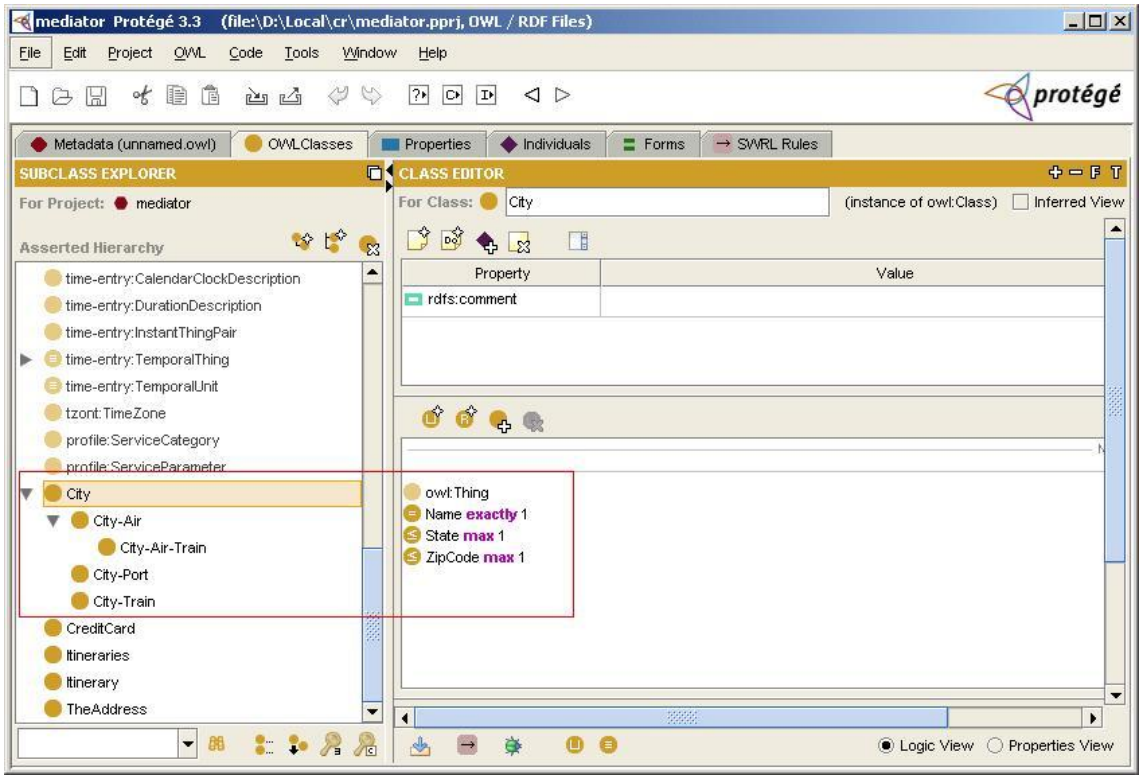


FIG. 6.9 – The Concepts definitions in Protege

- section 6.2.1 describes the federation architecture and the communication between the federated mediators,
- section 6.2.2 discusses the way the *Tell* and the *Ask* actions has been implemented,
- and section 6.2.3 concerns the OWL-S description of the federation.

6.2.1 Federation Management and Communication

In our mediator federation prototype system (see figure 6.1), heterogeneity is experimented thanks to three different knowledge representation languages (see figure 6.10) and knowledge interchange languages. We actually considered *(i)* KIF, which is designed for F-logic [44], *(ii)* DAML+OIL, which is designed for the DLs [52], and CGIF, defined for the CGs [64]. We also defined an exchange format for the $\mathcal{ALN}_{r+}$ which is founded on the extension of DAML+OIL. The mediators federation defines the common syntax formats as Resource Description Framework (RDF) description documents [56], the RDF description documents being referenced by the syntax translators.

From the point of view of federation management, *mediators federation* is a decentralized service federation. There are not one manager role in the federation, which is a hybrid architecture (see figure 2.9). At a knowledge representation level, we opt the organization strategy. The *complement* of query will be firstly send to homogeneous KR mediators, when this mediator can't find a local full *satisfaction* for this query. The capability discovery action explores the federated mediators in a way of "near by near". If one mediator

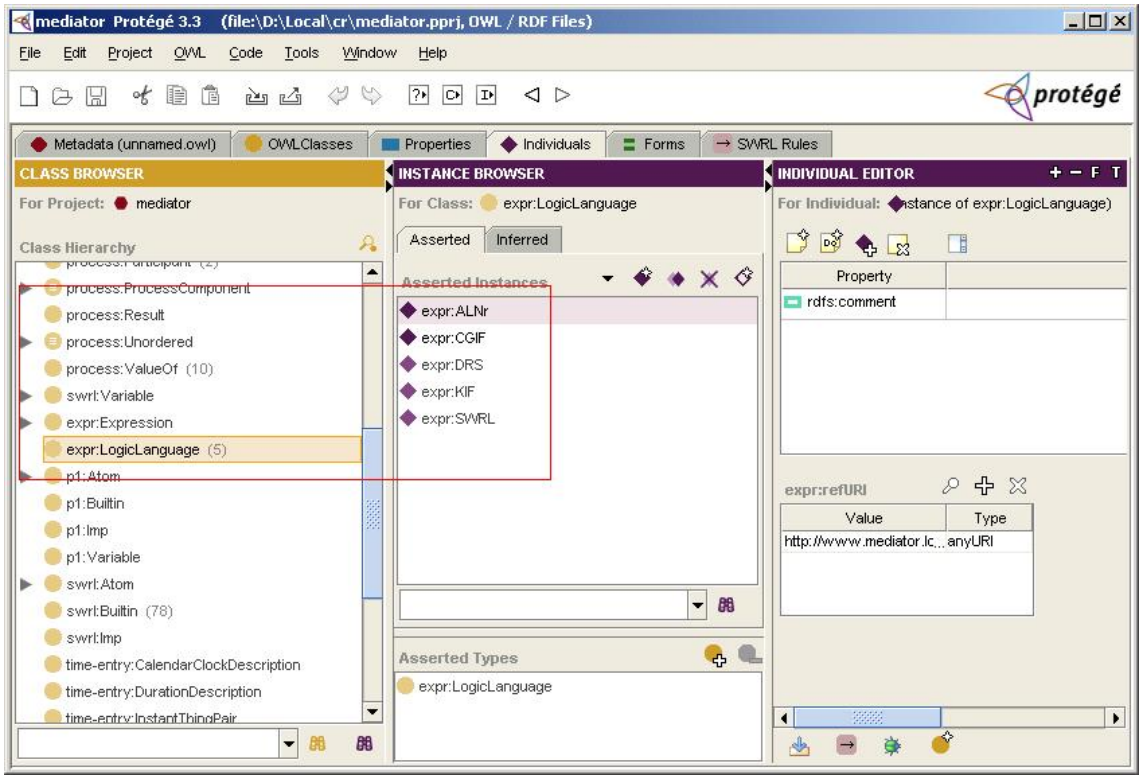


FIG. 6.10 – The Knowledge Representation Languages

understands two or more knowledge representations, then it will naturally provide a gateway service between the heterogeneous knowledge representations. The query will be sent to a known *mediator* and a selection will be done according to some quantitative criteria (number of peers, number of possible hops, etc.).

The mediator server is a kind of Web Service, i.e., that the mediators support the interoperable Machine to Machine interaction. The common terminologies are the assumption that there is also a machine readable description of the operations supported by the server, a description in the Web Services Description Language (WSDL). So we write a WSDL description for the mediator servers (see figure 6.11).

The WSDL of mediator service describes how to communicate using the mediator services. The WSDL description can be understood by the other mediator server, which describe the service ports of mediator. The WSDL is imported in *Protégé* in figure 6.12. We see that it define two service ports : *Tell* and *Ask*. The ports is defined by associating the mediator server address with a reusable service binding. In the description, the collection of ports define the mediator service. In the prototype system, it opts the Simple Object Access Protocol (SOAP) [94] for the communication between the mediator servers. The SOAP messages in this WSDL of mediator server, TellSoapIn, TellSoapOut, AskSoapIn, and AskSoapOut, describe the data being exchanged, and port types are abstract collections of supported operations.

The two principal actions, *Tell* and *Ask*, are defined as operations on the mediator service (see figure 6.12).

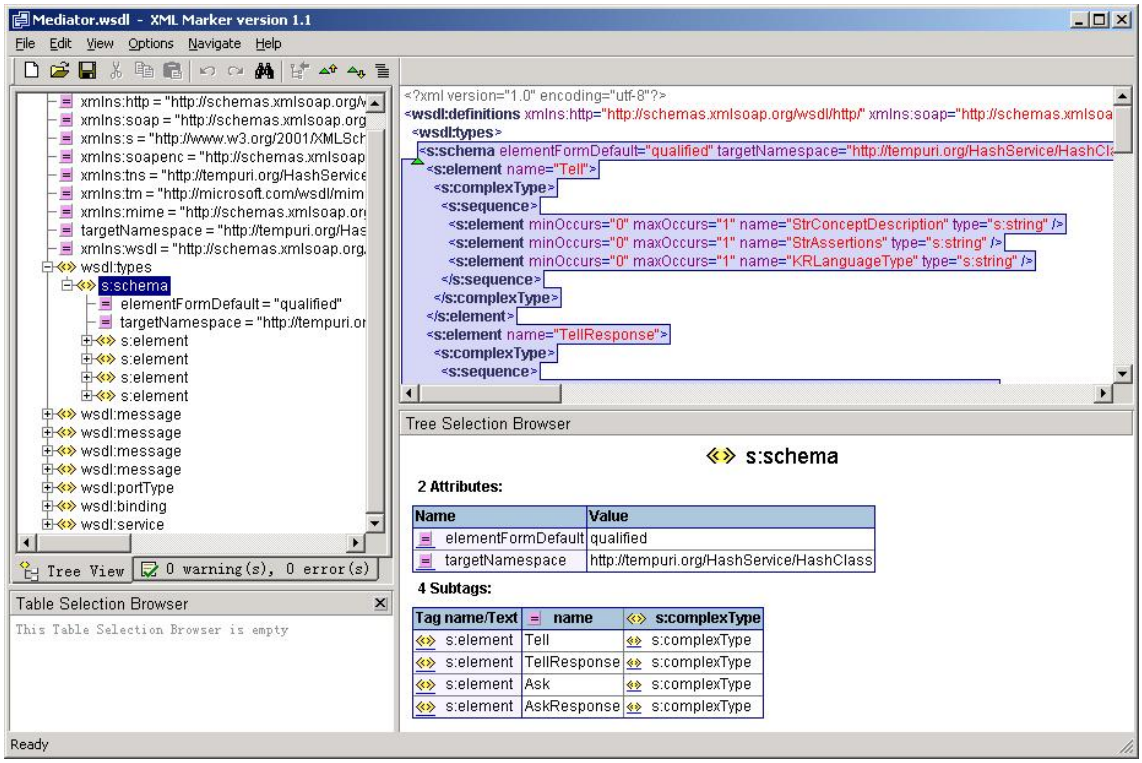


FIG. 6.11 – The WSDL Description of Mediator

6.2.2 *Tell* and *Ask* as SOAP Messages

The *Tell* operation has a simple result which is a boolean value. The *Tell* operations succeeds, when the result is true, and it has three input parameters : *StrConceptDescription*, *StrAssertions*, and *KRLanguageType*, which are defined as :

```
<s :element name="Tell">
  <s :complexType>
    <s :sequence>
      <s :element minOccurs="0" maxOccurs="1"
        name="StrConceptDescription" type="s :string" />
      <s :element minOccurs="0" maxOccurs="1"
        name="StrAssertions" type="s :string" />
      <s :element minOccurs="0" maxOccurs="1"
        name="KRLanguageType" type="s :string" />
    </s :sequence>
  </s :complexType>
</s :element>
```

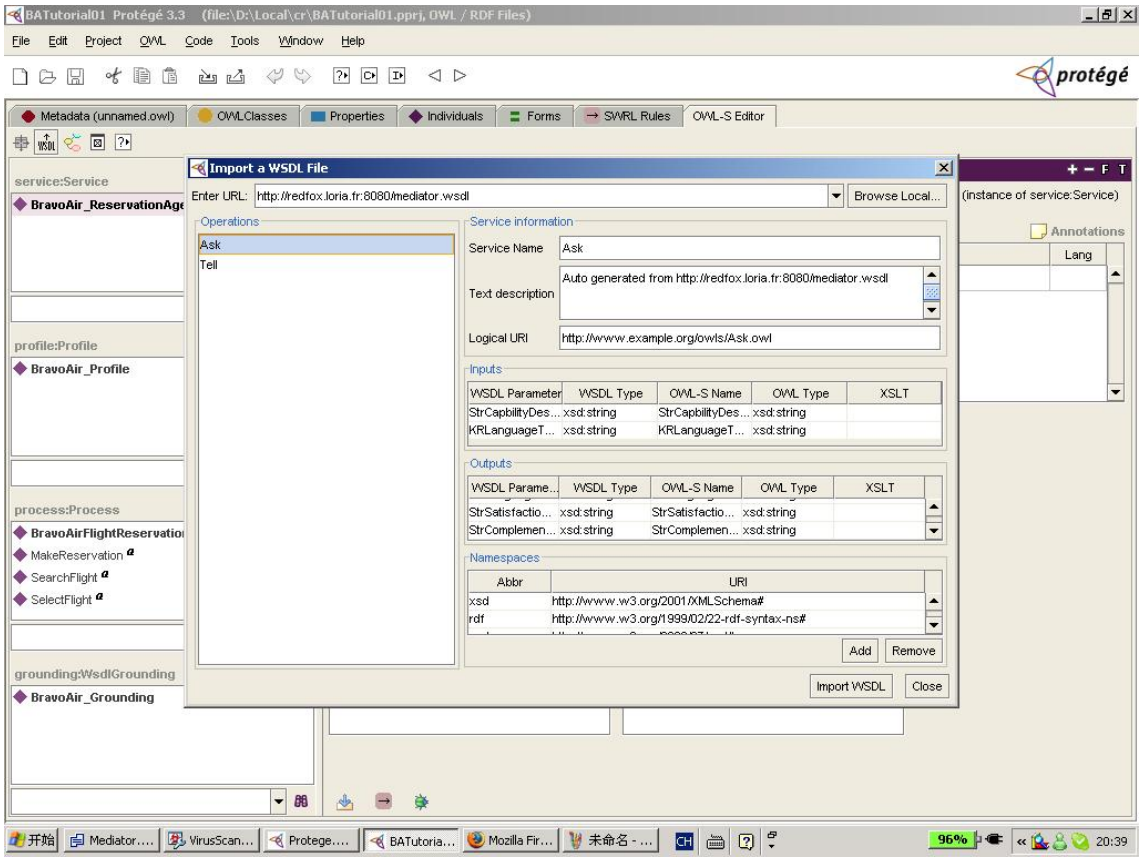


FIG. 6.12 – The imported WSDL Description

The *Ask* operation has two parameters : *StrCapbilityDescription* and *KRLanguage-Type*, which are defined as :

```
<s :element name="Ask">
  <s :complexType>
    <s :sequence>
      <s :element minOccurs="0" maxOccurs="1"
        name="StrCapbilityDescription" type="s :string" />
      <s :element minOccurs="0" maxOccurs="1"
        name="KRLanguageType" type="s :string" />
    </s :sequence>
  </s :complexType>
</s :element>
```

The result of an *Ask* operation is a composite answer which is described as a complex type having the following structure :

```
<s :element name="AskResponse">
  <s :complexType>
    <s :sequence>
      <s :element minOccurs="0" maxOccurs="1"
        name="StrSatisfactionDescription" type="s :string" />
      <s :element minOccurs="0" maxOccurs="1"
        name="StrComplementDescription" type="s :string" />
      <s :element minOccurs="0" maxOccurs="1"
        name="KRLanguageType" type="s :string" />
    </s :sequence>
  </s :complexType>
</s :element>
```

Some parameters, like *StrSatisfactionDescription* in *AskResponse* and *StrCapability-Description* in *Ask*, are capability/concept descriptions in one kind of knowledge representation language. These types of parameters, viewed as SOAP messages, serves for transforming a capability description from one mediator language (KIF, for example) into an other mediator language (CGIF, for example).

The prototype implements a complete mediator server in Description Logic, where the *StrSatisfactionDescription* and *StrCapabilityDescription* are in $\mathcal{ALN}_{r+}$, and they are noted in standard OWL. The SOAP messages carry the descriptions to other mediator servers. The SOAP messages of the *Ask* action example of `AIR-PORT` are :

SOAP *Ask* :

```
<SOAP-ENV :Envelop
...>
  <SOAP-ENV :Body>
    <s :Ask xmls :m="http ://mediator/namespaces">
      <StrCapabilityDescription>
        <owls :IntersectionOf>
          <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
          <owls :IntersectionOf>
            <owls :ObjectRestriction
                owls :property="http ://mediator/mediation.daml#has-ferry">
              <owls :someValuesFrom>
                <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
              </owls :someValuesFrom>
            </owls :ObjectRestriction>
            <owls :ObjectRestriction
                owls :property="http ://mediator/mediation.daml#has-flight-">
              <owls :someValuesFrom>
                <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
              </owls :someValuesFrom>
            </owls :ObjectRestriction>
          </owls :IntersectionOf>
        </owls :IntersectionOf>
      </StrCapabilityDescription>
      <KRLanguageTypes>ALNr</KRLanguageTypes>
    </s :Ask>
  </SOAP-ENV :Body>
</SOAP-ENV :Envelop>
```

The *Ask* SOAP message includes the two elements, *StrCapabilityDescription* and *KRLanguageType*. The value of *KRLanguageType* is `OWL` that means the *StrCapabilityDescription* is in OWL language. The *StrCapabilityDescription* is a OWL class objet corresponding to the description `(and CITY (∀ has-flight CITY) (∀ has-ferry CITY))`.

As we showed in the chapter 5, the description `(and CITY (∀ has-flight CITY))` is satisfied, and the `(∀ has-ferry CITY)` is the *complement*. The response to the *Ask* looks like :

SOAP *AskResponse* :

```
<SOAP-ENV :Envelop
...>
  <SOAP-ENV :Body>
    <s :AskResponse xmls :m="http ://mediator/namespaces">
      <StrSatisfactionDescription>
        <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
        <owls :IntersectionOf>
          <owls :ObjectRestriction
              owls :property="http ://mediator/mediation.daml#has-flight-">
            <owls :someValuesFrom>
              <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
            </owls :someValuesFrom>
          </owls :ObjectRestriction>
        </owls :IntersectionOf>
      </owls :IntersectionOf>
    </StrSatisfactionDescription>
    <StrComplementDescription>
        <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
        <owls :IntersectionOf>
          <owls :ObjectRestriction
              owls :property="http ://mediator/mediation.daml#has-ferry">
            <owls :someValuesFrom>
              <owls :Class owls :name="http ://mediator/mediation.daml#CITY"/>
            </owls :someValuesFrom>
          </owls :ObjectRestriction>
        </owls :IntersectionOf>
      </owls :IntersectionOf>
    </StrComplementDescription>
    <KRLanguageTypes>ALNr</KRLanguageTypes>
  </s :AskResponse>
 </SOAP-ENV :Body>
</SOAP-ENV :Envelop>
```

In the two SOAP messages, we note $\mathcal{ALN}_{r+}$ in the OWL language, the standard ontology description language for Web Services. An other extend OWL, OWL-S [63], is used for the service descriptions.

6.2.3 OWL-S for Mediator Federation

Indeed, the composition of mediators is described in OWL-S [63]. The OWL-S mediators federation describes the properties and the capabilities of each mediator services in an unambiguous, computer-intepretable form. OWL-S markup of mediator services facilitates the automation of mediator service tasks, including automated mediator service

discovery, execution, composition and interoperation. In the prototype system, the mediators federation is described as an OWL-S document. Indeed, figure 6.13 shows the OWL-S description of the mediators federation prototype system in an OWL-S editor, a plug-in ware in *Protégé* [43] :

1. The left part of the window shows some services, operations, profiles and bindings.
2. The middle part of the window shows the sequence of operations (actally, the *Ask* operation sequence in this figure), built from items appearing in the left part of the window.
3. The right part of the window shows the diagram of the *Ask* operation sequence, a clearer version of the sequence diagram of the *Ask* operation being in figure 6.14. These types of diagrams are automatically generated by the OWL-S editor we plugged in *Protégé* starting from what is indicated in the middle part of the window.

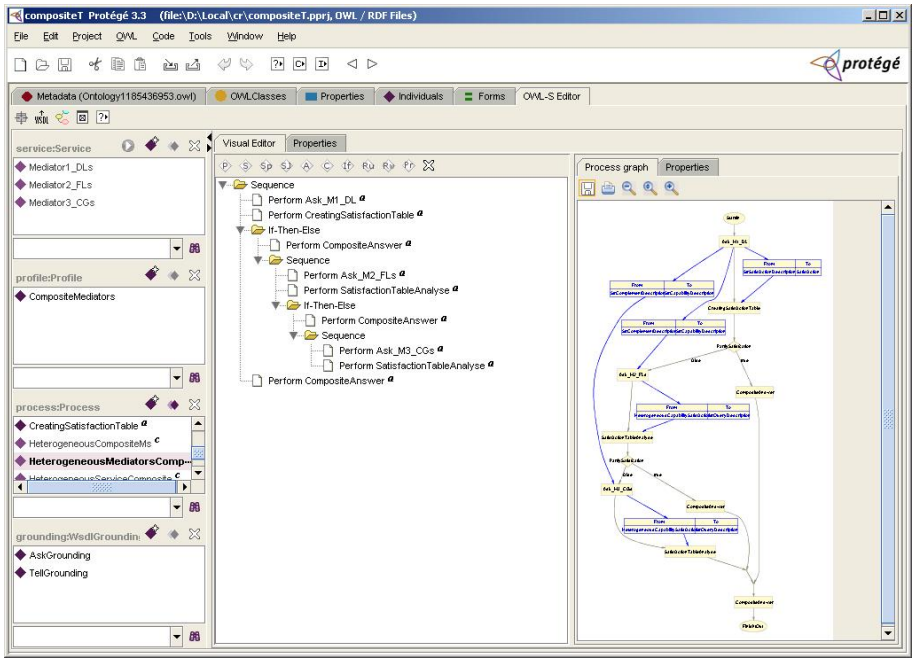


FIG. 6.13 – The Mediators Composition

6.3 Conclusion

This chapter presented and detailed the implementation of the mediator federation prototype system. This prototype consists of a complete mediator server supporting DLs, and two incomplete mediator servers in two different knowledge representations : F-Logic and CGs. The two incomplete mediator servers supports the capability discovery query action, and that is sufficient to implement a heterogeneous mediator federation environment.

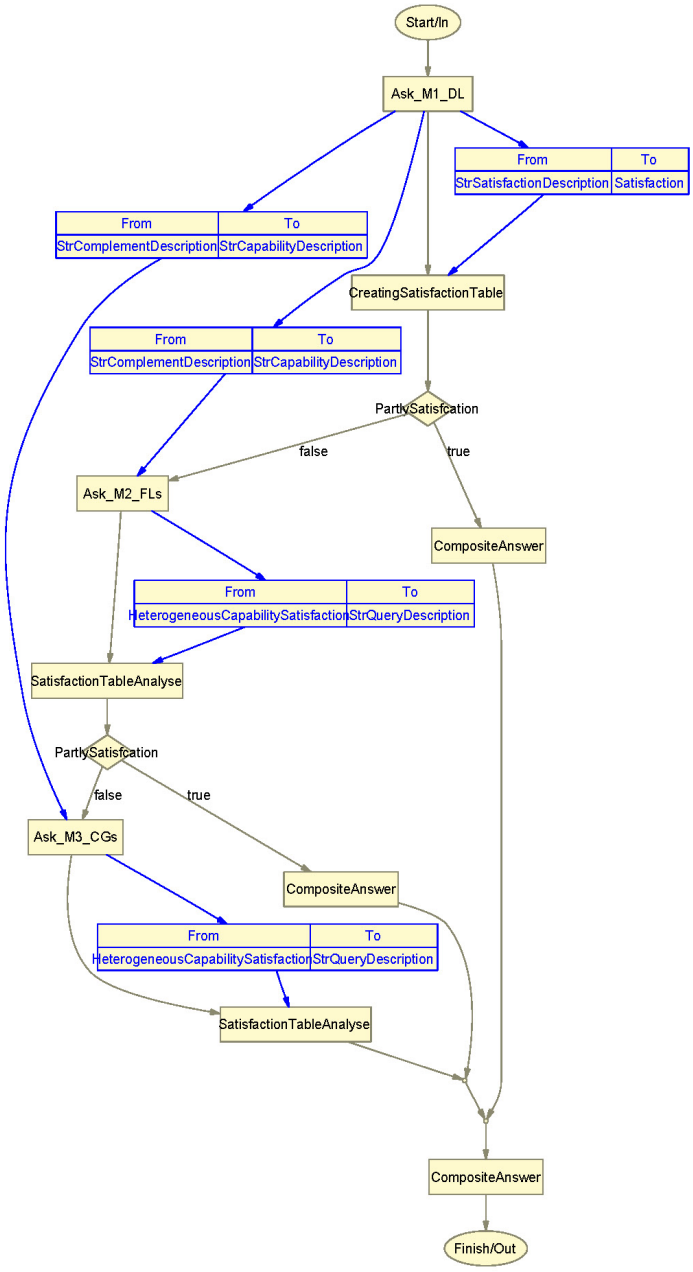


FIG. 6.14 – The Mediators Composition Detail

We have implemented the complete DL mediator server prototype in Java. We designed the mediator server prototype thanks to four program modules : *(i)* the local repository, *(ii)* the reasoning processor, *(iii)* the syntax translator, and *(iv)* the lexical ontological dictionary.

The local repository module is an implementation of a knowledge base in $\mathcal{ALN}_{r+}$, and it implements some main inference services, that includes the *subsumption test*, the *satisfaction calculation*, and the *complement calculation*. These inference services are sufficient to support capability management and discovery, even though some inferences (like *satisfiability testing*, and *disjointness testing* [6, 57]) are not implemented. These inferences can help the repository to provide a higher quality of service.

The reasoning processor works at the query context management and mediation federation management. This prototype depends on a manual OWL-S document to the query context management and the mediation federation management. UDDI[100] standard supports the any entities publish their service listings and discover each other and define how the services or software applications interact over the Internet. UDDI can apply in the mediation federation architecture to implement dynamic mediator service discovery.

The syntax translator supports the translation capability description structure $\forall R.A$ in three knowledge representation languages. We wrote the programs to implement the translation rules, so the programs must be rewritten whenever a new language is used in the mediation federation. We may develop a tool to automatically or semi-automatically describe the syntax translation rules.

The lexical ontological dictionary is an application programming interface for lexical mapping, using WordNet. There are many specific ontology dictionaries are developed for different domains as chemistry, medicine, and biology [82], etc.. The mediator server must manage and use multiple ontological dictionaries, that also must be studied in the ontology domain now.

The complete DLs mediator server can process the *Ask* and the *Tell* actions. In an *Ask* action, it can access the two heterogeneous mediator servers which are in *F-Logic* and *CGs*. The prototype implemented our design of capability management and discovery. On the other hand, this prototype also raise some new points on architecture, ontology, and etc., that is the frame of part of our future work, as more deeply treated in the final chapter of this thesis.

Chapitre 7

Conclusions

In this thesis work, we consider that two main results have been achieved :

1. defining a *capability* description language $\mathcal{ALN}_{r+}$ to support capability management and inference services ;
2. designing a mediation federation system in order to implement the composite answer approach for capability discovery in heterogeneous knowledge representation environments.

Considering the first result and regarding the objectives of this thesis, as outlined in chapter 1 (section 1.2.2, page 32) :

1. *From the capability description and organization perspective* : $\mathcal{ALN}_{r+}$ relies on the characteristics of *roles* (i.e. relationships between *concepts*) for entity's capability description. $\mathcal{ALN}_{r+}$ inherits and extends the knowledge representation theory and technology of Description Logic. We implemented some inference services which may be used for capability management and capability discovery. We also introduced an open model of restriction f to describe the relationship between *roles* (i.e. the relationship between relationships of *concepts*). We introduced a very simple relationship description model f in the capability description language $\mathcal{ALN}_{r+}$. In this thesis work, we detailed several relationships between capabilities by the description model f, which includes *subsumption*, *composition*, and *equivalence*. And then we analyzed the relationships between the relationships of capabilities, that is not a complete analysis result. The model f is an open description model, which can describe other relationships between capabilities. Nevertheless, more complex relationships between the relationships of capabilities may exist, and *modal logic* and/or *second-order logic* theories may be used in the description and the inference on the relationships between the relationships among capabilities.
2. *From the capability discovery perspective* : we rely on a single concept (the *complement concept*) and on its procedural determination to satisfy some functionalities that are useful for capability, competence as well as knowledge management. Namely, these facilities concern :
   (a) *the comparison of intentionally defined individuals*, i.e. two sets of individuals characterized by their $\mathcal{ALN}_{r+}$ formula, can be compared thanks to the provided mechanisms : they are considered as "equal", in terms of capabilities, when

one formula *exactly satisfies* the second one ; one is considered as "greater than the other", always in terms of capabilities, when one formula *subsumes* the other one (cf. the notion of *wider satisfaction*) and so on.

(b) *the identification of capability gaps* : this is clearly achieved thanks to the identification of the complement,

(c) *filling up capability gaps* : this is attained thanks to the complement calculation,

(d) *constraining the set of candidates* that may contribute to an identified gap filling-up : this is the role of the strategy that may be imposed on the complement calculation process.

Considering the second result, we have designed and implemented a mediation federation system prototype. The prototype is fully written in Java programming language (Sun Microsystem's JDK1.2 and Java Web Services Developer Pack 1.6), and it has been successfully verified on the following operating systems : Miscrosoft Windows (95, 98, Me, XP, 2000, 2003) and Linux (Red Hat 7.X and Mandrake 8.x). The proposed prototype has a mediation architecture based on standard Web Services.

However, the whole mediator federation prototype is under a hybrid system architecture. An importer, an exporter and a mediator server compose a typical SOA system, but the mediator federation is a pure P2P architecture. As compared to the P2P system's capabilities that were introduced in the chapter 2 (section 2.3.2 on page 55), our prototype has the following characteristics :

1. *Regarding autonomy* : this is clearly achieved thanks to the fact that every mediator server can solely provide the services (*Ask* and *Tell*) to its client (importer and exporter) ;
2. *Regarding dynamicity* : This prototype depends on a manual OWL-S document to manage the whole mediator federation : that cannot work in an application system. UDDI [100] standard supports how entities publish their list of services and discover each other, and it defines how the services or software applications interact over the Internet. UDDI principle can be applied in the mediator federation architecture to implement dynamic mediator service management and discovery. However, this property of the federation requires additional mechanisms for the actual management of the federation i.e. mechanisms for dynamically enabling a mediator to enter or to leave (temporarily or permanently) an existing federation. This aspect was not addressed in this work since it was not considered as central for our needs.
3. *Regarding decentralization* : The architecture we propose is clearly decentralized since an importer can address its query to any mediator in the federation.
4. *Regarding cooperation* : This is also a characteristics of our architecture since, when a local repository cannot satisfy a query requirement, the local mediator can cooperate with heterogeneous partners for the calculation of a composite answer.

From an other standpoint, considering the characteristics of mediators, as introduced in chapter 3 (section 3.1.3, page 63) :

1. The prototype does not provide sophisticated *context management* facilities. It implements some simple composite answer strategies, like *satisfaction first*, *at most*

x, and so on. Currently, dealing with these types of strategies requires additional programming efforts : the provision of a generic strategy engine requires further investigations.

2. Based on the simple context management, the *query propagation* is based on a "proximity" criteria (as opposed to a semantic criteria). The queries are always sent to the closest mediator as an OWL-S document description.

Therefore, dynamic federation management, context management and semantic criteria for query propagation and how they interact need an across-the-board design of mediators federation in the future.

In addition, based on the conceptual models and architectures we proposed, we have a prototype system of mediation federation. It completes a lot of the validity checking work on the theories and approaches in this thesis. However, we obviously need to enrich the syntax translator to support more knowledge representation systems. On the other hand, the capability discovery service can be applied in many domains, like the implementation of UDDI or dynamic ERPs.

In an other way forward, we think also about the capability application at two additional research directions and applications : *(i)* capability representation theory and implementation and *(ii)* applying the prototype system in some actual application domains.

Firstly, $\mathcal{ALN}_{r+}$ is not a sufficiently rich syntax concept description language. For example, it does not support the full negation and concept disjunction, which will bring on some NP complete problems. Therefore, we intend to explore description languages which supports more complex *concept* and *role* descriptions, and then to provide complete semantic inference algorithms. As a path toward this aim, the expressive power of description Logics as a representation language of *capabilities* and *entities* is more limited than graph-based languages. And we intend to examine the capability application based on the graph-based knowledge representation technology. It is well-known that the approaches and algorithms of graph-based knowledge representation always face the NP problem, but we think that the description of the relationships between capabilities should help in creating more powerful algorithms and approaches for graph-based knowledge representations.

Secondly, the mediator federation architecture faces some communication problems that are similar to those encountered in other heterogeneous distributed architectures, like Web Services in W3C, CORBA in OMG, and RM-ODP in ISO. In this thesis, we defined a *capability description* concept for query communication between mediators : for more general applications, we need a kind of "standard query language" which abstractly describes the syntax of capability discovery queries. This standard must then be implemented and/or supported by almost all knowledge representation systems.

Annexe A

Example from the INTEROP Knowledge Map

During this doctoral study I participated into the INTEROP Network of Excellence (IST-508 011, http ://www.interop-noe.org). I contributed to the conceptual design of the repository, called Knowledge Map, that will serve for building and maintaining Knowledge about a given domain, namely the enterprise software and applications interoperability domain. I give a capability discovery example on the INTEROP Knowledge Map in this section, starting from its conceptual design.

The whole INTEROP Knowledge Map is sketched in the UML class diagram in figure A.1 which references to the INTEROP Knowledge Map Requirements report [11].

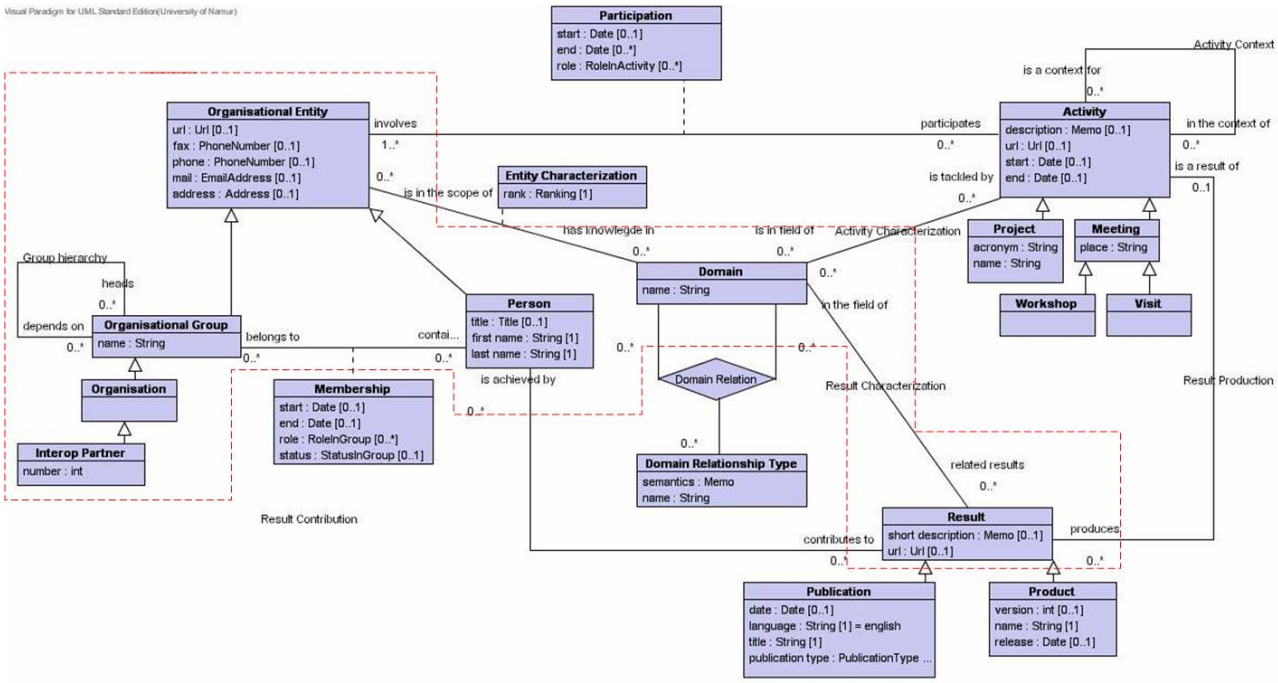


FIG. A.1 – The UML class diagram of The INTEROP Knowledge Map

The UML classes and associations we will mention in the example, are inside the dotted-line circle in figure A.1. These classes and associations are described in the $\mathcal{ALN}_{r+}$ knowledge base.

1. The UML *classes* correspond to the *concept* descriptions in $\mathcal{ALN}_{r+}$ that are represented in figure A.2.
2. The UML *generalization* corresponds to the *subsumption relationship*, and the UML *associations* correspond the *role* descriptions. The *role* descriptions are called *capability* descriptions in this work and they are presented in figure A.3.
3. We show some *individual data* of the knowledge map in figure A.4, these individual data will be used further.

These three parts are representatives of the INTEROP Knowledge Map, when viewed as a knowledge base in $\mathcal{ALN}_{r+}$. The knowledge base can support the capability discovery query.

Looking for some persons or some organizations have the skill in a domain, that is very frequent query to the INTEROP knowledge map. The query can be written in *capability discovery structure* as $\forall$`skill`.$\top$.

The `skill` does not exist in the $\mathcal{T}r_{Interop}$, so the `skill` is a foreign term to the knowledge base of INTEROP Knowledge Map. Ahead, we calculate the similar term of `skill` in $\mathcal{T}r_{Interop}$, the algorithm has been mentioned in the previous sections. The results are shown on the figure A.5.

In the similarity calculation results, the similarity value of `skill` with `knowledge` is the largest (0.83), that mean the `knowledge` is the most similar semantic with `skill` in the similarity calculation algorithm. So the `knowledge` is a similar term for `skill` in $\mathcal{T}r_{Interop}$.

On the other hand, the role `has-knowledge-in` is subsumed by `knowledge` in $\mathcal{T}r_{Interop}$, so `has-knowledge-in` possibly satisfies the `skill` too by the definition 5.1.4 (page 94). There are two possible satisfactions (`has-knowledge-in` and `knowledge`) to the `skill` in $\mathcal{T}r_{Interop}$.

Further, a query always need to find some entity objects to practically satisfy a requirement. For example, an INTEROP research task need to build an enterprise ontology, that require some entities which has skill in `enterprise modeling` and `ontology`. The capability discovery describes the query as : $\forall$`skill`.$\top$(`enterprise modeling ; ontology`).

By the previous mentions, the role `has-knowledge-in` possibly satisfy the `skill`. There are some assertions of `has-knowledge-in` in $\mathcal{A}_{Interop}$. It does not exist an individual has both two skills in $\mathcal{A}_{Interop}$, so a composite answer is calculated.

$$
\begin{aligned}
\text{Ask :} \quad \mathcal{L}_{Query}(Q) &= \forall\texttt{skill}.\top(\texttt{enterprise modeling};\texttt{ontology}) \\
&\times \\
KB_{Interop} &= \{\mathcal{T}_{Interop}, \mathcal{T}r_{Interop}, \mathcal{A}_{Interop}\} \\
&\rightarrow \\
\mathcal{L}_{Answer}(Q) &= \{\texttt{has-knowledge-in(UHP/LORIA, ontology)}, \\
&\quad \texttt{has-knowledge-in(David.Chen, enterprise modeling)}\}
\end{aligned}
$$

$$
\begin{array}{lrcl}
C_1 & \texttt{Entity} & \dot{\sqsubseteq} & \top \\
C_2 & \texttt{Abstract} & \dot{\sqsubseteq} & \top \\
C_3 & \texttt{Domain} & \doteq & \texttt{(and Abstract} \\
 & & & \quad \texttt{(all name Abstract))} \\
C_4 & \texttt{Organisational-Entity} & \doteq & \texttt{(and Entity} \\
 & & & \quad \texttt{(some url Entity)} \\
 & & & \quad \texttt{(some fax Entity)} \\
 & & & \quad \texttt{(some phone Entity)} \\
 & & & \quad \texttt{(some mail Entity)} \\
 & & & \quad \texttt{(some address Entity)} \\
 & & & \quad \texttt{(some has-knowledge-in Domain))} \\
C_5 & \texttt{Organisational-Group} & \doteq & \texttt{(and Organisational-Entity} \\
 & & & \quad \texttt{(all name Organisational-Entity)} \\
 & & & \quad \texttt{(some contain Person))} \\
C_6 & \texttt{Organisation} & \dot{\sqsubseteq} & \texttt{Organisational-Group} \\
C_7 & \texttt{Interop-Partner} & \doteq & \texttt{(and Organisation} \\
 & & & \quad \texttt{(all number Organisation))} \\
C_8 & \texttt{Result} & \doteq & \texttt{(and Entity} \\
 & & & \quad \texttt{(some url Entity)} \\
 & & & \quad \texttt{(some short-description Entity)} \\
 & & & \quad \texttt{(some in-the-field-of Domain))} \\
C_9 & \texttt{Person} & \doteq & \texttt{(and Organisational-Entity} \\
 & & & \quad \texttt{(some title Organisational-Entity)} \\
 & & & \quad \texttt{(all first-name Organisational-Entity)} \\
 & & & \quad \texttt{(all last-name Organisational-Entity)} \\
 & & & \quad \texttt{(some contributes-to Result))}
\end{array}
$$

FIG. A.2 – An $\mathcal{ALN}_{r+}$-TBox from INTEROP Knowledge Map ($\mathcal{T}_{Interop}$)

$$
\begin{array}{lrcl}
R_1 & \texttt{knowledge} & \dot{\sqsubseteq} & \top_{role} \\
R_2 & \texttt{contributes} & \dot{\sqsubseteq} & \top_{role} \\
R_3 & \texttt{is} & \dot{\sqsubseteq} & \top_{role} \\
R_4 & \texttt{has-knowledge-in} & \dot{\sqsubseteq} & \texttt{knowledge} \\
R_5 & \texttt{contributes-to} & \dot{\sqsubseteq} & \texttt{contributes} \\
R_6 & \texttt{in-the-field-of} & \dot{\sqsubseteq} & \texttt{is}
\end{array}
$$

FIG. A.3 – An $\mathcal{ALN}_{r+}$-T$_r$Box from INTEROP Knowledge Map ($\mathcal{T}r_{Interop}$)

```
Interop-Partner(UHP/LORIA), Person(David.Chen),
Interop-Partner(University of Bordeaux 1),
Organisation(University of Edinburgh),
has-knowledge-in(UHP/LORIA, ontology),
has-knowledge-in(University of Edinburgh, ontology),
has-knowledge-in(David.Chen, enterprise modeling),
```

FIG. A.4 – An $\mathcal{ALN}_{r+}$-ABox from INTEROP Knowledge Map ($\mathcal{A}_{Interop}$)

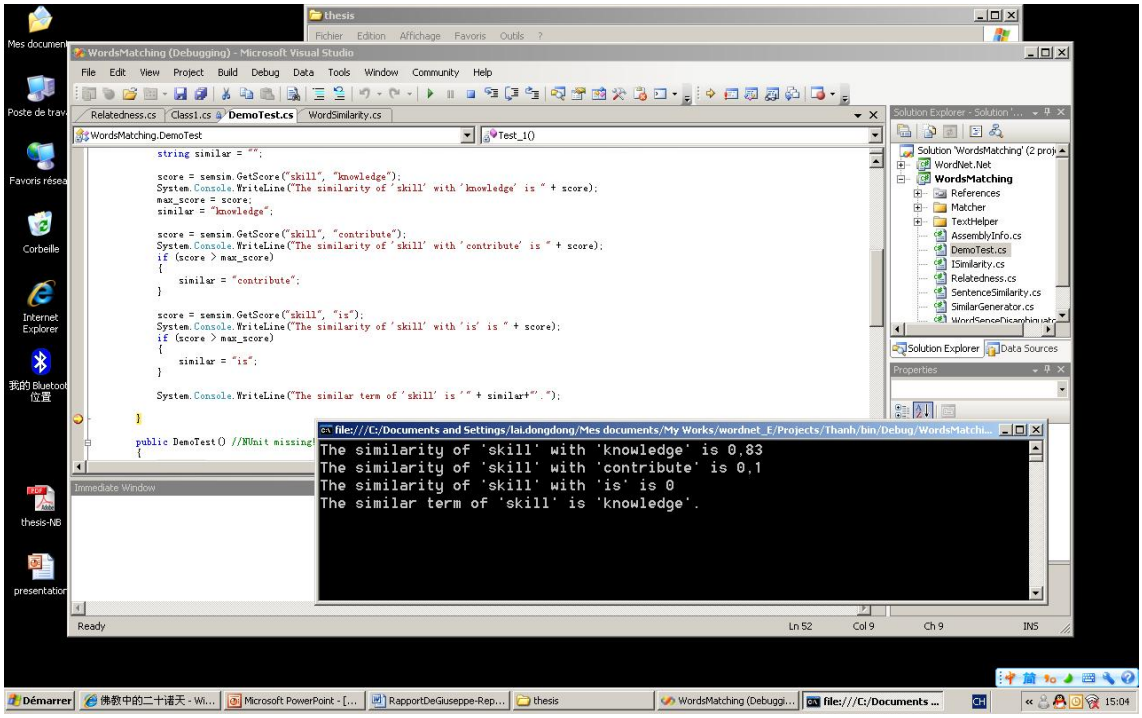


FIG. A.5 – The calculation of similar term

The composite answer $\mathcal{L}_{Answer}(Q)$ is calculated in similar approaches which have been detailed in the previous sections. By the INTEROP Knowledge Map example, we can see the capability discovery approaches can be applied in many knowledge discovery systems. These approaches can well work in the heterogeneous knowledge representation environment only supported by some current lexical dictionaries.

Bibliographie

[1] N. Boudjlida A. Benna and H. Talantikite.
Sawsdl, mediation and xquery for web services.
In D. Benslimane and A. Ouksel, editors, *Proceedings of the 8th international conference on New Technologies in Distributed Systems : Notere 2008*, volume 1, pages 23–27, Lyon France, June 2008. ACM edition.

[2] Karl Aberer, Magdalena Punceva, Manfred Hauswirth, and Roman Schmidt.
Improving data access in p2p systems.
IEEE Internet Computing, 6(1) :58–67, 2002.

[3] Kathleen Ahrens.
When love is not digested : Underlying reasons for source to target domain pairings in the contemporary theory of metaphor.
In Yuchau E.Hsiao, editor, *Proceedings of the Congnitive Linguistics Conference*, pages 273–302, 2002.

[4] Kathleen Ahrens, Siaw Fong Chung, and Chu-Ren Huang.
Conceptual metaphors : Ontology-based representation and corpora driven mapping principles.
In Mark Lee John Barnden, Sheila Glasbey and Alan Wallington, editors, *Proceedings of the ACL 2003 Workshop on the Lexicon and Figurative Language*, pages 35–41, 2003.

[5] Gustavo Alonso and Fabio Casati.
Web services and service-oriented architectures.
In *ICDE*, page 1147. IEEE Computer Society, 2005.

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

[7] Franz Baader, Ralf Küsters, and Ralf Molitor.
Structural subsumption considered from an automata-theoretic point of view.
In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Description Logics : Proceedings of the 1998 International Workshop on Description Logics (DL'98), IRST, Povo - Trento, Italy, June 6-8, 1998*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

[8] Franz Baader, Ralf Küsters, and Ralf Molitor.
Computing least common subsumers in description logics with existential restrictions.
In Thomas Dean, editor, *IJCAI : Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 96–103. Morgan Kaufmann, 1999.

[9] Jon Barwise.
Heterogeneous reasoning.
Lecture Notes in Computer Science, 699 :64–74, 1993.

[10] C. Beeri, A. Levy, and M-C. Rousset.
Rewriting Queries Using Views in Description Logics.
In *ACM Symposium on Principles Of Database Systems*, pages 99–108, Tucson, Arizona, 1997.

[11] Khalid Benali and all.
Interop work package 1 : Knowledge map requirements.
Technical report, UHP/LORIA, March 2004.

[12] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu.
Data management for peer-to-peer computing : A vision.
In *WebDB*, pages 89–94, 2002.

[13] Elisa Bertino, Mauro Negri, Giuseppe Pelagatti, and Licia Sbattella.
Object-oriented query languages : The notion and the issues.
IEEE Transactions on Knowledge and Data Engineering, 4(3) :223–237, June 1992.

[14] A. Borgida.
Description Logics in Data Management.
IEEE Transactions on Knowledge and Data Engineering, 7(5) :671–682, 1995.

[15] A. Borgida and P. Devanhu.
Adding more "DL" to IDL : Towards more Knowledgeable Component Interoperability.
In *21rst International Conference on Software Engineering, ICSE'99*, pages 378–387, Los Angeles, CA, May 1999. ACM Press.

[16] A. Borgida and D. Etherington.
Hierarchical Knowledge Bases and Efficient Disjunctive Reasoning.
In *Proc. First International Conference on Principles of Knowledge Representation and Reasoning*, pages 33–43, Toronto, May 1989.

[17] Alexander Borgida and Peter F. Patel-Schneider.
A semantics and complete algorithm for subsumption in the classic description logic.
J. Artif. Intell. Res. (JAIR), 1 :277–308, 1994.

[18] M. Bouchikhi and N. Boudjlida.
Using Larch to Specify the Behavior of Objects in Open Distributed Environments.
In *Proceedings of the 1998 Maghrebian Conference on Software Engineering and Artificial Intelligence*, pages 275–287, Tunis, Tunisia, December 1998.
98-R-300.

[19] N. Boudjlida.
Knowledge in Interoperable and Evolutionary Systems.
In L. Dreschler-Fischer and S. Pribbenow, editors, *KRDB'95, Workshop on "Reasoning about Structured Objets : Knowledge Representation Meets Databases"*, pages 25–26, Bielefeld, Germany, September 1995.
(Position Paper).

[20] N. Boudjlida.
A Mediator-Based Architecture for Capability Management.
In M.H. Hamza, editor, *Proceedings of the 6th International Conference on Software Engineering and Applications, SEA 2002*, pages 45–50, MIT, Cambridge, MA, November 2002.

[21] R. J. Brachman.
A Structural Paradigm for Representing Knowledge.
PhD thesis, Harvard University, 1977.

[22] Ronald J. Brachman.
Taxonomy, descriptions, and individuals in natural language understanding.
In *ACL*, 1979.

[23] Ronald J. Brachman and Hector J. Levesque.
The tractability of subsumption in frame-based description languages.
In *AAAI*, pages 34–37, 1984.

[24] Ronald J. Brachman and James G. Schmolze.
An overview of the KL-ONE knowledge representation system.
Cognitive Science, 9(2) :171–216, 1985.

[25] D. Calvanese, D. de Giacomo, M. Lenzerini, D. Nardi, and R. Rosati.
Information Integration : Coceptual Modeling and Reasoning Support.
In *6th International Conference on Cooperative Information Systems, CoopIS'98*, pages 280–291, 1998.

[26] Dong Cheng and Boudjlida Nacer.
Federated mediators for query composite answers.
In *6th International Conference on Enterprise Information Systems - ICEIS'2004, Porto, Portugal*, volume 4, pages 170–175, Apr 2004.

[27] Christopher W. Clifton and Wen-Syan Li.
Semint : A tool for identifying attribute correspondences in heterogeneous databases using neural networks.
Data and Knowledge Engineering, 33(1), April 2000.

[28] Simona Colucci, Stefano Coppi, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Agnese Pinto, and Azzurra Ragone.
Semantic-based resource retrieval using non-standard inference services in description logics.
In Andrea Calì, Diego Calvanese, Enrico Franconi, Maurizio Lenzerini, and Letizia Tanca, editors, *SEBD : Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems, SEBD 2005, Brixen-Bressanone (near Bozen-Bolzano), Italy, June 19-22, 2005*, pages 232–239, 2005.

[29] L. Palopol D. Sacc and D. Ursino.
Semi-automatic, semantic discovery of properties from database schemes.
In *Proceedings of the 1998 International Symposium on Database Engineering & Applications*, page 244. IEEE Computer Society, 1998.

[30] J. Dennis and E. Van Horn.
Programming semantics for multiprogrammed computations.
Communications of the Association of Computing Machinery, pages 143–155, March 1966.

[31] Premkumar Devanbu and Eric Wohlstadter.
Evolution in distributed heterogeneous systems, November 27 2001.

[32] DL-org.
Description logics, 2007.
http//dl.kr.org/.

[33] AnHai Doan, Pedro Domingos, and Alon Y. Levy.
Learning source description for data integration.
In *WebDB (Informal Proceedings)*, pages 81–86, 2000.

[34] C. Dong.
Capability representation on increased $\mathcal{ALN}$.
In *Doctoral symposium of Interoperability for Enterprise Software and Applications Conference*, Bordeaux, France, March 2006.

[35] F. M. Donini, M. Lenzerini, and D. Nardi.
The complexity of existential quantification in concept languages.
Artificial Intelligence, 53(2-3) :309–327, February 1992.

[36] T. Dyck.
Uddi 2.0 provides ties that bind, April 2002.
(http ://www.eweek.com/).

[37] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, and Rukman Senanayake.
The OWL-S editor - A development tool for semantic web services.
In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC : The Semantic Web : Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, volume 3532 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2005.

[38] Enrico Franconi.
Natural language processing.
In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook : Theory, Implementation, and Applications*, pages 450–461. Cambridge University Press, 2003.

[39] Brian Gaines.
Organizational Knowledge Acquisition, volume Handbook on Knowledge Management 1 of *International Handbooks on Information Systems*, chapter 16.
Springer, Knowledge Science Institute, University of Calgary, Calgary, AB, Canada T2N 1N4, 2003.
ISBN :978-3-540-43527-3.

[40] Hector Garcia-Molina et al.
The TSIMMIS approach to mediation : Data models and languages.
Journal of Intelligent Information Systems, 8(2) :117–132, 1997.

[41] Michael R. Genesereth.
Knowledge Interchange Format, 1998.

[42] M. Geneserith.
Knowledge interchange format.
In J. Allen, editor, *Proceedings of the 2nd International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 238–249. Morgan Kaufman, 1991.

[43] Gennari, John H., Musen, Mark A., Fergerson, Ray W., Grosso, William E., Monica Crubezy, Henrik Eriksson, Noy, Natalya F., Tu, and Samson W.
The evolution of protege : an environment for knowledge-based systems development.
International Journal of Human-Computer Studies, 58(1) :89–123, 2003.

[44] Matt Ginsberg.
Knolwedge interchange format : The KIF of death.
Technical report, Matt Ginsford, 1991.
(AI Magazine).

[45] Wiederhold Gio.
Mediators in the architecture of future information systems.
Computer, 25(3) :38–49, March 1992.

[46] Mounira Harzallah Giuseppe Berio.
Knowledge management for competence management.
Journal of Universal Knowledge Management, 0(1) :21–28, June 2005.

[47] P. Gochet and P. Gribomont.
Logique, volume 1.
Hermès, Paris, 1991.

[48] T. R. Gruber.
Toward principles for the design of ontologies used for knowledge sharing.
International Journal of Human-Computer Studies, 43 :907–928, November 1995.

[49] T-D. Han, S. Purao, and V. Storey.
A Methodology for Building a Repository of Object-Oriented Design Fragments.
In *18th International Conference on Conceptual Modelling, ER'99*, pages 203–217, Paris, November 1999. Spriger Verlag.
LNCS 1728.

[50] T. Hoppe, C. Kindermann, J. J. Quantz, A. Schmiedel, and M. Fischer.
Back V5—Tutorial & manual.
KIT-Report 100, Technische Universität, Berlin, 1993.

[51] I. Horrocks.
Description Logic : Axioms and Rules.
Talk given at Dagstuhl "Rule Markup Technique" Workshop, February 2002.
http ://www.cs.man.ac.uk/ horrocks/Slides/.

[52] Ian Horrocks.
DAML+OIL : a description logic for the semantic web.
IEEE Data Engineering Bulletin, 25(1) :4–9, 2002.

[53] M. Kifer and J. Wu.
A logic for object-oriented logic programming (maier's O-logic : Revisited).
In *ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, Philadelphia*, March 1989.

[54] Michael Kifer, Georg Lausen, and James Wu.
Logical foundations of object-oriented and frame-based languages.
Journal of the Association for Computing Machinery, 42 :741–843, July 1995.

[55] Judith Klavans and Min-Yen Kan.
Role of verbs in document analysis.
In *COLING-ACL*, pages 680–686, 1998.

[56] Jacek Kopecký and Bijan Parsia.
Web services description language (WSDL) version 2.0 : RDF mapping.
World Wide Web Consortium, Working Draft WD-wsdl20-rdf-20070326, March 2007.

[57] Ralf Küsters.
Non-Standard Inferences in Description Logics, volume 2100 of *Lecture Notes in Computer Science*.
Springer, 2001.

[58] Maurizio Lenzerini and Andrea Schaerf.
Concept languages as query languages.
In *AAAI*, pages 471–476, 1991.

[59] T. Lipkis.
A KL-ONE classifier.
In J. G. Schmolze and R. J. Brachman, editors, *Proceedings of the KL-ONE Workshop, Jackson (NH), USA*, BBN Report 4842/Fairchild Technical Report 618, pages 126–143, 1982.

[60] R. MacGregor and R. Bates.
The LOOM knowledge representation language.
Technical Report ISI/RS-87-188, Information Science Institute, University of Southern California, Marina del Rey (CA), USA, 1987.

[61] Robert MacGregor and Mark H. Burstein.
Using a description classifier to enhance knowledge representation.
IEEE Expert, 6(3) :41–46, June 1991.

[62] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm.
Generic schema matching with cupid.
In *The VLDB Journal*, pages 49–58, 2001.

[63] David Martin and all.
Owl-s : Semantic markup for web services.
W3C Recommendation, November 2004.
http ://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.

[64] Philippe Martin.
Knowledge representation in CGLF, CGIF, KIF, frame-CG and formalized-english.
In Uta Priss, Dan Corbett, and Galia Angelova, editors, *ICCS, Conceptual Structures : Integration and Interfaces, 10th International Conference on Conceptual Structures, ICCS 2002, Borovets, Bulgaria, July 15-19, 2002, Proceedings*, volume 2393 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2002.

[65] George A. Miller and al.
Wordnet 2.0, 2003.
http ://www.cogsci.princeton.edu/ wn/.

[66] Tova Milo and Sagit Zohar.
Using schema matching to simplify heterogeneous data translation.
In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 122–133. Morgan Kaufmann Publishers Inc., 1998.

[67] P. Mitra, G. Wiederhold, and J. Jannink.
Semi-automatic integration of knowledge sources.
In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

[68] Prasenjit Mitra, Gio Wiederhold, and Martin Kersten.
A graph-oriented model for articulation of ontology interdependencies.
Lecture Notes in Computer Science, 1777 :86+, 2000.

[69] Ralf Möller and Volker Haarslev.
Description logics for the semantic web : Racer as a basis for building agent systems.
KI, 17(3) :10, 2003.

[70] Michael Moore and Tatsuya Suda.
Adaptable peer-to-peer discovery of objects that match multiple keywords.
In *SAINT Workshops*, pages 402–407. IEEE Computer Society, 2004.

[71] A. Napoli.
Une introduction aux logiques de description.
Technical Report RR No 3314, INRIA-LORIA, Nancy, 1997.

[72] N.Boudjlida and C. Dong.
Enterprise Informations Systems, volume VI, chapter Federated Mediators for Query Composite-Answers, pages 31–38.
Kluwer Academics Publishers, ISBN10 1-4020-3674-4, 2006.

[73] Bernhard Nebel.
Teminological reasoning is inherently intractable (research note).
Artificial Intelligence, 43(2) :235–249, May 1990.

[74] Ian Niles and Adam Pease.
Linking lexicons and ontologies : Mapping wordnet to the suggested upper merged ontology.
In *Proceedings of the IEEE International Conference on Information and Knowledge Engineering*, pages 412–416, 2003.

[75] Ikujiro Nonaka and Hirotaka Takeuchi.
The Knowledge - Creating Company : How Japanese Companies Create the Dynamics of Innovation.
Oxford University Press, Oxford, 1995.

[76] OMG.
The Object Model Architecture Guide.
Technical Report 91.11.1, Revision 2.0, Object Management Group, September 1992.

[77] Maurice Pagnucco and Norman Y. Foo.
Inverting resolution with conceptual graphs.
In Guy W. Mineau, Bernard Moulin, and John F. Sowa, editors, *ICCS, Conceptual Graphs for Knowledge Representation, ICCS '93, Quebec City, Canada, August 4-7, 1993, Proceedings*, volume 699 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 1993.

[78] Luigi Palopoli and al.
A unified graph-based framework for deriving nominal interscheme properties type conflicts and object cluster similarities.
Fourth IECIS International Conference on Cooperative Information Systems, pages 34–45, September 02 - 04 1999.

[79] Massimo Paolucci and Katia P. Sycara.
Autonomous semantic web services.
IEEE Internet Computing, 7(5) :34–41, 2003.

[80] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman.
Medmaker : A mediation system based on declarative specifications.
In *Proceedings of the 12th International Conference on Data Engineering (ICDE '96)*, pages 132–141, Washington - Brussels - Tokyo, February 1996. IEEE Computer Society.

[81] Christine Parent and Stefano Spaccapietra.
Issues and approaches of database integration.
Communications of the ACM, 41(5es) :166–178, May 1998.

[82] Ted Pedersen, Serguei V. S. Pakhomov, Siddharth Patwardhan, and Christopher G. Chute.
Measures of semantic similarity and relatedness in the biomedical domain.
Journal of Biomedical Informatics, 40(3) :288–299, 2007.

[83] Christof Peltason.
The BACK system - an overview.
SIGART Bulletin, 2(3) :114–119, 1991.

[84] Vossen Piek.
Eurowordnet : A multilingual database with lexical semantic networks.
EuroWordNet General Document, 3, July 19 1999.

[85] H. Pinto, A. Prez, and J. Martins.
Some issues on ontology integration.
In *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods(KRR5)*, pages In [18], 7–1 – 7–12, 1999.

[86] WIKIMEDIA project.
Wikipedia, 2008.
http//www.wikipedia.org/.

[87] Joachim Quantz and Carsten Kindermann.
Implementation of the BACK-system version 4.
Technical Report TUB-FB13-KIT-78, KIT Project Group Publications, September 1 1990.
Tue, 07 Nov 1995 19 :43 :08 GMT.

[88] Erhard Rahm and Philip A. Bernstein.
A survey of approaches to automatic schema matching.
VLDB Journal : Very Large Data Bases, 10(4) :334–350, September 2001.

[89] Howard Shrobe Randall Davis and Peter Szolovits.
What is a knowledge representation ?
AI Magazine, 14(1) :1993, 1993.

[90] G. A. Ringland and D. A. Duce.
Approaches to knowledge representation : an introduction (ed.).
Letchworth, Hertfordshire, England : Research Studies Press New York : Wiley, c1988, 260 p. (Knowledge-based and expert systems series) CALL NUMBER : QA76.76.E95 A669 1988, 1988.

[91] M. Schmidt-Schauss and G. Smolka.
Attribute Concepts Description with Complements.
Artificial Intelligence Journal, 48(1) :1–26, 1991.

[92] Manfred Schmidt-Schauß ; and Gert Smolka.
Attributive concept descriptions with complements.
Artificial Intelligence, 48(1) :1–26, 1991.

[93] Valeria De Antonellis Silvana Castano.
Global viewing of heterogeneous data sources.
IEEE Transcation on Knowledge and Data Engineering, 13(2) :277–297, April 2001.

[94] Simple object access protocol (SOAP) 1.1.
W3C Note, May 2000.

[95] J. F. Sowa.
Conceptual graphs for a database interface.
IBM Journal of Research and Development, 20, July 1976.

[96] John F. Sowa.
Syntax, Semantics, and Pragmatics of Contexts.
In Gerard Ellis, Robert Levinson, William Rich, and John Sowa, editors, *LNAI 954, Conceptual Structures : Applications, Implementation and Theory*, pages 1–15. Springer-Verlag, Berlin, 14–18 August 1995.
Proceedings of the 3rd Int. Conf. on Conceptual Structures, (ICCS'95), Santa Cruz, CA, USA.

[97] John F. Sowa.
Knowledge Representation : logical, philosophical, and compuational foundations.
Brooks/Cole, Pacific Grove, CA, 2000.

[98] SUO.
The ieee standard upper ontology web site, 2003.
http ://suo.ieee.org.

[99] UDDI Consortium.
UDDI Technical White Paper, September 2000.
http ://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF.

[100] UDDI.org.
Uddi : Universal description, discovery and integration.
Technical White Paper, September 2003.
(http ://uddi.org).

[101] Jeffrey D. Ullman.
Information integration using logical views.
In *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece, January 1997. Springer, Berlin.

[102] P. Velardi, A. Cucchiarelli, and M. Petit.
Supporting scientific collaboration in a network of excellence through a semantically indexed knowledge map.
In G. Morel G. Doumeingts, J. Muller and B. Vallespir, editors, *Proceedings of the 2nd International Conference On Interoperability for Entreprise Software and Applications*, pages 231–241, Bordeaux, France, March 2006. I-ESA'2006, Springer Verlag.

[103] Arjan Vernhout.
Management challenges in the competence-based organization.
Competence-based Management Website, 2007.

[104] W3C.
Semantic Web, 2003.
http ://www.w3.org/2001/sw.

[105] W3C.
Web Ontology, 2003.
http ://www.w3.org/2001/sw/WebOnt.

[106] W3C.
Web Services, 2003.
http ://www.w3.org/2002/ws.

[107] Gerhard Jurgen Wickler.
Using Expressive and Flexible Action Representation to Reason about Capabilities for Intelligent Agent Cooperation.
PhD thesis, University of Edinburgh, 1999.

[108] Gio Wiederhold.
Mediators in the architecture of future information systems.
IEEE Computer, 25(3) :38–49, March 1992.

[109] Gio Wiederhold.
Mediators in the architecture of future information systems.
In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[110] Zhibiao Wu and Martha Stone Palmer.
Verb semantics and lexical selection.
In *ACL*, pages 133–138, 1994.