République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.
Université A-Mira de Béjaïa Faculté des Sciences Exactes
Département Informatique

ÉCOLE DOCTORALE RÉSEAUX ET SYSTÈMES DISTRIBUÉS

Mémoire de Magistère

En Informatique

Option : Réseaux et Systèmes Distribués

Thème

Etude de l'Utilisation des Ontologies pour une Organisation d'une Carte de Connaissance dans un Processus de Data Mining

---

Présenté par

Hayette KHALED

Devant le jury composé de

Président	M^{r} KERKAR Moussa	Professeur	U. A/Mira Béjaïa.
Rapporteur 1	M^{r} KECHADI M-Tahar	Professeur	U. College Dublin.
Rapporteur 2	M^{r} TARI A-Kamel	M.C.A	U. A/Mira Béjaïa.
Examinatrice 1	M^{me} BENATCHBA Karima	Professeur	E.S.I, Alger.
Examinatrice 2	M^{me} NADER Fahima	M.C.A	E.S.I, Alger.
Invité	M^{r} DAHAMNI Foudil	M.A.A	E.S.I, Alger.

Béjaïa, 2011

Remerciements

Louange Á Dieu, le miséricordieux, sans Lui
rien de tout cela n'aurait pu être.

Je *remercie tout d'abord mes encadreurs, Professeur M-Tahar KECHADI et Docteur A-Kamel TARI pour l'attention et le suivi qu'ils ont porté à mon travail, leurs conseils et commentaires précieux qui m'ont permis de surmonter mes difficultés et de progresser dans mon projet.*

J'*adresse mes sincères remerciements au Professeur Moussa KERKAR, Professeur Karima BENATCHBA, Docteur Fahima NADER ainsi qu'au Docteur Foudil DAHAMNI qui m'ont honoré par leurs jugements du mon travail.*

Mes *remerciements les plus chaleureux vont vers mes chers parents qui m'ont tout appris et tout donné ainsi vers mes frères Abdenour et Walid, mes soeurs Saliha, Sabah et son mari Rafik, Salima et son mari Mohammed et leur fille Ikrame que j'adore, Zakia, Soufia et Souria pour leurs aides et accompagnements, je leur dis tous je vous aime et dieu vous protège pour moi vous êtes mon trésor.*

Mon *coeur Noureddine, je te remercie pour ton amour et ta tendresse avec lesquels tu m'a toujours entouré ce qui m'a permis ainsi d'oublier le stress et la fatigue du travail.*

J'*adresse ma reconnaissance, particulièrement, au Professeur Farid Meziane source des articles IEEE pour sa simplicité et sa disponibilité.*

Je *dédie ce travail particulièrement à* M^{me} *Malika YAICI qui a été et restera toujours mon exemple.*

Je *n'oublie pas de remercier tous mes amis de l'école doctorale RESYD ainsi que ma copine Hayate WAHCHI, je leur souhaite tous la réussite dans leur vie.*

Comme *les meilleurs restent pour la fin, je dédie mon travail à la mémoire de ma chère Grand-mère et mon ange Amina.*

AYA

Dédicaces

A *Mes très chers parents,*

A *Mon ange Ikrame,*

A *Mes adorables frères et soeurs,*

A *Mes beaux-frères et leurs familles,*

A *Mon amour Noureddine et sa famille,*

A *Tous mes amis et collègues.*

Résumé

Dans le domaine de Data Mining, d'énormes collections de données réparties sont traitées et en résulte des connaissances dont la gestion est très difficile surtout dans le cas où elles appartiennent à différentes organisations. En littérature, peu de techniques sont proposées dans ce contexte et nous soulignons que la technique la plus répondue est l'architecture Knowledge Map (KM) qui se base sur des répertoires de méta-connaissances construits manuellement par des simples utilisateurs. Pour remédier à ce problème, nous proposons une architecture Knowledge Map à base Ontologique (KMO). La solution proposée s'inspire de KM, elle automatise la construction des répertoires de méta-connaissances en utilisant des ontologies, et est distribuée selon le Tree P2P (TreeP).

L'architecture KMO est constituée de cinq composants à savoir : le KM local, le noyau de KM, le navigateur de KM, le retriever de KM et le KO Manager. Le noyau de KM contient les répertoires de méta-connaissances.

Nous avons développé un algorithme de construction de répertoires de méta-connaissances (ayant pour entrées les ontologies et les connaissances à organiser) ainsi que les algorithmes de recherche de connaissances et ceux pour les mises à jour de l'architecture. Nous avons implémenté l'algorithme de construction de répertoires et celui de la recherche de connaissances. La validation de notre approche a été faite par le calcul de la complexité des algorithmes proposés qui sont logarithmiques, montrant ainsi la robustesse de notre solution.

Mots clés **:** Data Mining (DM), Knowledge Map (KM), Tree P2P (TreeP),Knowledge Map à base Ontologique (KMO).

Abstract

In data mining field, massive collections of distributed data are treated, and this leads to have knowledge that is difficult to manage, especially in the case where they belong to different organizations. In literature, a few techniques are suggested in this context, and we highlight that the technique that is well-spread is the Knowledge Map (KM) architecture, which is based on repositories of meta-knowledge that are manually constructed by ordinary users. To solve this problem, we suggest an Ontology-Based Architecture for Knowledge (KMO). The proposed solution is inspired from KM ; it automates the construction of meta-knowledge repositories by using Ontologies. The distribution of this architecture is done according to Tree P2P (TreeP).

KMO architecture consists of five components, namely local KM, KM core, KM navigator, KM retriever and KO Manager. The KM core contains repositories of meta-knowledge.

We have developed an algorithm for constructing meta-knowledge repositories (Having the Ontologies and the knowledge to organize as input) as well as the search algorithms of knowledge and those used to update the architecture.

We have implemented the algorithm of the construction of the repositories and the one related to the research of Knowledge. The validation of our approach has been done by calculating the complexity of the proposed algorithms that are logarithmic, showing the robustness of our solution.

Keywords **:** Data Mining (DM), Knowledge Map (KM), Tree P2P (TreeP),Knowledge Map Ontology based (KMO).

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Liste des abréviations

API	**A**pplication **P**rogramming **I**nterface
BDD	**B**ases **D**e **D**onnées
DAS	**D**ata **A**ccess **S**ervice
DDM	**D**istributed **D**ata **M**ining
DM	**D**ata **M**ining
EPMS	**E**xecution **P**lan **M**anagement **S**ervice
I-MIN	**I**ntension **MIN**ing
KBR	**K**nowledge **B**ase **R**epository
KCs	**K**nowledge **C**oncentrates
KDD	**K**nowledge **D**iscovery and **D**ata mining
KDS	**K**nowledge **D**irectory **S**ervice
KEPR	**K**nowledge **E**xecution **P**lan **R**epository
K-Grid	**K**nowledge Grid
KM	**K**nowledge **M**ap
KMO	**K**nowledge **M**ap **O**ntology
KMR	**K**nowledge **M**etadata **R**epository
KO Manager	**K**nowledge **O**ntology **M**anager
OWL	**O**ntology **W**eb **L**anguage
OWL DL	**O**ntology **W**eb **L**anguage **D**escription **L**ogics
RAEMS	**R**esource **A**llocation and **E**xecution **M**anagement **S**ervice
RDF	**R**esource **D**escription **F**ramework
RDFS	**R**esource **D**escription **F**ramework **S**chema
RPS	**R**esults **P**resentation **S**ervice
TAAS	**T**ools and **A**lgorithms **A**ccess **S**ervice

TMR **T**ask **M**etadata **R**epository

TreeP **Tree** **P**ear to **P**ear

URI **U**niform **R**esource **I**dentifier

W3C **W**orld **W**ide **W**eb **C**onsortium

XML **E**xtensible **M**arkup **L**anguage

Introduction générale

De nos jours, de grands volumes de données dans divers domaines sont rassemblés et stockés. L'extraction efficace de la connaissance à partir de ces données est une problématique clé en recherche. Ceci encourage le développement des techniques de Data Mining Distribuées (DDM) pour traiter ces collections de données énormes, multidimensionnelles et réparties sur un ensemble de sites distants ; ce qui crée un problème de gestion des résultats (appelés connaissances) qui devient encore plus critique lorsque les connaissances sur différents sites appartiennent à différentes organisations. Pour remédier à ce problème, une représentation efficace des connaissances extraites à partir de ces données est primordiale afin de guider l'utilisateur dans le processus de leur interprétation pour mieux exploiter le domaine qui l'intéresse.

Les techniques DDM existantes se basent sur l'exécution parallèle des données locales sur des nœuds individuels puis de générer les modèles globaux en faisant l'agrégation des résultats locaux. Une gestion efficace des connaissances distribuées est l'un des facteurs clés affectant les résultats de ces techniques.

Récemment, Le-Khac et al [1] ont proposé une architecture Knowledge Map (KM) pour le traitement d'un grand ensemble de connaissances distribuées. La première structure de KM est basée sur la topologie 1-n (1 serveur, n clients) qui est appropriée pour un niveau de distribution modéré. Ils ont déployé leur propre KM sur une autre topologie de réseau efficace, appelée Tree P2P (TreeP) [2] [3] sur de grands environnements distribués comme Grid/P2P. Cette architecture permet de rechercher les connaissances de manière rapide et avec une grande précision en se basant sur des répertoires de méta-connaissances [4]. Le système proposé facilite aussi le fusionnement et la coordination des résultats locaux pour générer des modèles globaux. Elle est l'une des couches principales de ADMIRE [5], un framework basé sur la plateforme Grille pour le développement des techniques DDM pour de grands ensembles de données hétérogènes et distribués.

Malgré tous les avantages que présentent cette architecture, ses concepteurs n'ont

pas encore généralisé le concept pour construire de manière automatique les répertoires de méta-connaissances qui constituent le composant clef de cette architecture et ils ne considèrent pas l'aspect sémantique de la connaissance.

Afin de résoudre ce problème, nous proposons l'architecture Knowledge Map à base d'Ontologies (KMO) qui exploite l'avantage d'expressivité des ontologies et leur sémantique pour construire les répertoires de méta-connaissances d'une manière automatique et d'améliorer ainsi la qualité des connaissances extraites. La distribution de cette architecture est faite selon le Tree P2P (TreeP) qui est une topologie d'arbre pour le P2P (pair à pair) car les ontologies sont structurées par nature en arbres qui ressemblent au TreeP. Nous avons développé un algorithme de construction des répertoires de méta-connaissances ayant pour entrées les ontologies et les connaissances à organiser. Les répertoires de méta-connaissances constituent le noyau de base de l'architecture KMO. Nous avons également développé des algorithmes de recherche des connaissances ainsi que ceux pour les mises-à-jour de l'architecture. Les algorithmes ont été implémentés sous JAVA et l'API JENA. Nous avons ensuite abordé la complexité de tous les algorithmes proposés. Leur complexité est d'ordre logarithmique et linéaire. Ce qui montre que cette nouvelle architecture est robuste et ainsi offre une amélioration dans le temps de création des répertoires de méta-connaissances. Cette architecture est très efficace et nécessaire dans le domaine de Data Mining où la quantité de connaissances est très grande et distribuée.

Le reste du mémoire est organisé en quatre chapitres, dans le premier chapitre nous présentons un état de l'art des techniques de représentation de connaissances et des ontologies. Le second chapitre détaille l'architecture Knowledge Map (KM). Dans le troisième chapitre nous présentons notre proposition Knowledge Map à base d'Ontologies (KMO). Le quatrième chapitre porte sur l'expérimentation et l'évaluation de notre architecture KMO. Finalement, nous terminons par une conclusion sur ce projet et quelques perspectives.

Chapitre 1

Techniques de Représentation de Connaissances et les Ontologies

1.1 Introduction

Le Data Mining (DM) ou fouille de données a attiré beaucoup d'attention dans tous les secteurs de l'économie ces dernières années, en raison d'énormes quantités de données disponibles et la nécessité de les convertir en information utile et en connaissances. Les informations et les connaissances acquises peuvent être utilisées comme outils d'aide à la décision.

Les systèmes d'information ont été témoin d'un chemin d'évolution dans le développement des fonctionnalités suivantes : la collecte de données et la création des entrepôts de données, la gestion des données et l'extraction des connaissances.

Des outils d'extraction des données sont développés pour permettre l'analyse de données et la découverte de modèles de données qui sont très intéressants. Les grandes quantités de données et l'hétérogénéité des modèles résultants nécessitent un système de gestion qui offre aux utilisateurs un accès rapide, assurant des résultats de bonne qualité ainsi qu'une utilisation efficace des connaissances. Ce système doit avoir un impact positif sur la phase de pré-traitement, il permettra aux utilisateurs d'avoir une réflexion sur les données à sélectionner, les techniques de DM à choisir ainsi que de la précision de la connaissance produite. Notre intérêt dans ce mémoire est la gestion et l'organisation des connaissances extraites par les techniques de DM ; ce que nous appelons phase de post traitement et non pas l'extraction de connaissances. Cette phase a connu un intérêt en recherche et une bibliographie très riche est disponible dans la littérature [6].

Le présent chapitre est organisé en quatre sections. Dans la section suivante nous

abordons la représentation des connaissances puis nous présentons quelques techniques de gestion de connaissances dans la section (1.3). La section (1.4) est consacrée à la synthèse sur ces techniques de gestion de connaissances. En fin, dans la dernière section nous présentons les ontologies et les outils qu'elles offrent.

1.2 Représentation des Connaissances

En intelligence artificielle, la connaissance de surface, basée sur l'expérience (le savoir-faire) se distingue de la connaissance profonde, liée au savoir. Le savoir est un ensemble de connaissances plus ou moins systématisées, acquises par une activité mentale suivie. Le savoir-faire est un ensemble de connaissances, d'expériences et de techniques accumulés par une personne ou une société [7].

La représentation des connaissances revient à établir une correspondance entre le monde extérieur et un formalisme symbolique qui peut être traité par un ordinateur. Le domaine de la connaissance est trop vaste et varié pour être représenté et exploité par un formalisme unique. De ce fait plusieurs représentations sont utilisées.

Nous reprenons la classification de J.L Laurière donnée dans [7]. Les connaissances sont classées par rapport à leurs natures de représentation : procédurales ou déclaratives. La figure (FIG. 1.1) [7] présente les divers formalismes allant du procédural (plus figé, plus structuré) au plus déclaratif (plus ouvert, plus libre).

La représentation déclarative permet de spécifier un savoir indépendamment des contraintes et des méthodes d'utilisation. Elle permet de répondre à une question de type *"quoi"*. La structure de contrôle est séparée des connaissances entrées sous formes de règles données en vrac. L'intérêt primordial est la modularité.

La représentation procédurale permet de traiter des problèmes de type algorithmique c'est-à-dire complètement analysés et entièrement connus. Elle exprime un flot d'informations structurées et traduit *"comment"* transite la connaissance qui est une simulation du comportement réel.

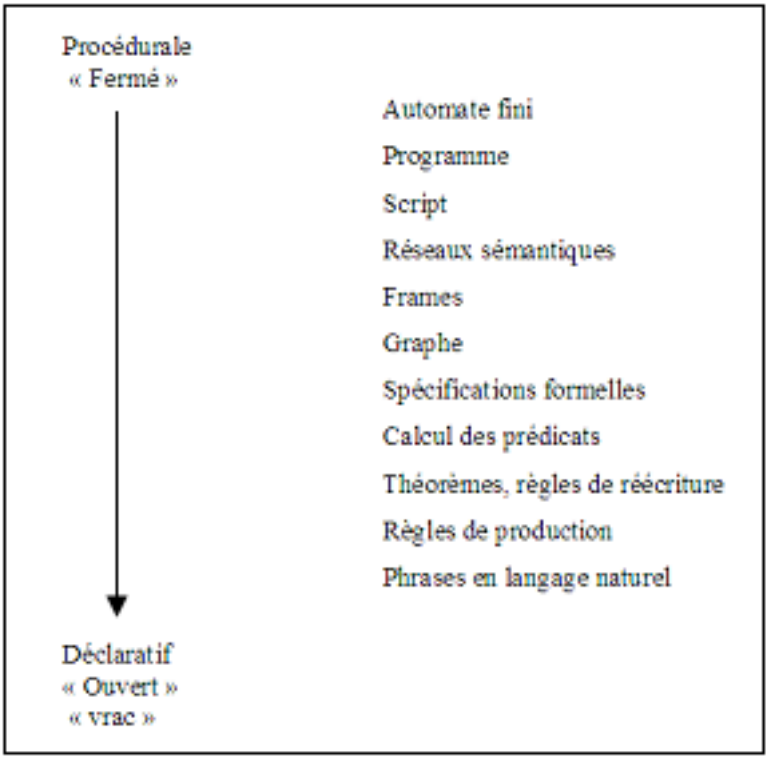

FIG. 1.1 – Classification de J.L. Laurrière.

Les représentations peuvent être regroupées dans les trois grandes classes suivantes :

- Procédurales : Incluant les automates finis et les programmes ;
- Déclaratives : Comprenant le calcul des prédicats et les règles de production
- Structurées : Réseaux sémantiques, frames, schémas, scripts, objets.

Certains modèles de représentation sont plus aptes à modéliser une connaissance fortement déductible, d'autres conviennent mieux à une connaissance descriptive et enfin d'autres encore à une connaissance structurée.

✍ Règles de productions [7]

Ce modèle de représentation est très répandu. Il est proche de la formulation naturelle des raisonnements et il est donc facile à utiliser. Les règles de production permettent de représenter des connaissances dynamiques. La syntaxe de représentation des règles est la suivante : *SI Prémisse(s) ALORS Conséquence(s).*

Par le fait que la conséquence d'une règle peut être prémisse d'une autre, l'ensemble des règles de production est organisé en réseau. Ce type de représentation permet de résoudre des problèmes de nature déductive et inductive.

✍ Réseaux sémantiques

Les réseaux sémantiques sont des graphes orientés sans cycle dont les noeuds représentent des objets, événements ou concepts et les arcs représentent des relations de tout type entre les noeuds.

Cette technique représente le sens des mots, elle est à base de l'interprétation du langage naturel. Les propriétés et les attributs se transmettent de noeud en noeud en glissant le long du réseau comme des associations d'idées. Ils permettent d'exprimer sous forme de liens et de gérer des relations entre objets hiérarchisés (héritage de propriétés, traitement d'exceptions). Ils permettent de représenter des connaissances statiques et dynamiques. La représentation graphique facilite la compréhension lorsqu'il s'agit pour l'utilisateur d'examiner le contenu de la base de connaissances.

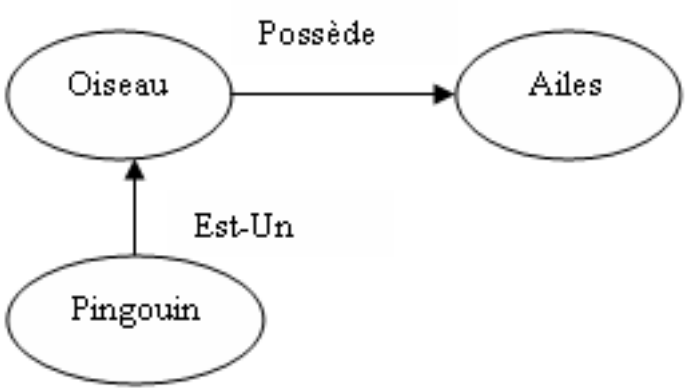

FIG. 1.2 – Réseau sémantique.

Il est aisé d'interpréter le schéma d'un réseau sémantique (voir FIG. 1.2) [8], nous pouvons en effet, y remarquer plusieurs éléments :

- Un pingouin est un oiseau ;
- Un oiseau possède des ailes ;

Par conséquent, nous pouvons dériver qu'un pingouin possède des ailes. En effet, un réseau sémantique possède les caractéristiques suivantes :

- Il s'agit d'une représentation selon un graphe acyclique ;
- Nous pouvons procéder à certaines inférences : en effet, il existe des relations prédéfinies telles que les relations :
  - *Est_ Un :* Il s'agit de l'inclusion ensembliste qui autorise le mécanisme d'héritage : une sous classe hérite de tous les liens sémantiques de la classe parent ;
  - *Instance_ de :* Il s'agit du lien entre une instance et la classe à laquelle elle appartient. Une instance partage tous les liens sémantiques de la classe à laquelle elle

appartient.

✍ Objets structurés

Le concept d'objet structuré est générique. Il recouvre les frames, les scripts, etc.

Frame : Le concept Frame était proposé par Minsky [9] pour guider l'interprétation d'une scène en vision par ordinateur. Un Frame est une structure de données incluant des informations à la fois déclaratives et procédurales, il représente une description d'un contexte composé d'une collection de rôles et de relations ce qui donne une description explicite.

Scripts : Les scripts ont été utilisés dans certains systèmes expérimentaux [10] pour la compréhension du langage naturel pour représenter les scénarios (ensemble d'évènements) avec la chronologie standard. Un script est composée de :

1. Une scène ;
2. Des props (Les objets manipulés dans le script) ;
3. Les acteurs (Les agents) ;
4. Les événements : Un changement de situation ;
5. Les situations : Une configuration des entités et relations.

Le script est une structure de situations. Dans chaque situation, un ou plusieurs acteurs font les actions. Les acteurs agissent dans les lieus de la scène avec les props.

Exemples d'un script :

1. Manger dans une bonne restaurant française
   Scène : La salle, l'entrée, les tables (comme lieu), la cuisine ;
   Acteurs : Le maître d'Hôtel, Le serveur, la "bus-boy", les clients ;
   Props : Le menu et la table (comme objet), les chaises, les couteaux, fourchettes et cuillers, les verres ;
   Situations : Entrer, S'asseoir, Lire le menu, Commander, Manger, Boire, Demander l'addition, Payer, Partir.

2. Achat d'une boisson au distributeur automatique
   Scène : Devant la machine ;
   Props : La machine, les pièces de monnaie, la boisson, le gobelet ;
   Acteur : Acheteur ;
   Situations : Sortir tes pièces de monnaie, payer, Sélectionner la boisson et les options

(sucre, crème, etc.), recouper la boisson et la monnaie.

✍ Logiques formelles

La logique formelle est un langage symbolique issu des logiques mathématiques permet de formuler des descriptions sous une forme proche du langage courant et représentable dans un langage de programmation. Les représentations logiques sont à la base de systèmes d'I.A.

✍ Les réseaux neuronaux

Les réseaux de neurones formels sont une abstraction des neurones physiologique [11] leur modélisation est donnée en figure (FIG. 1.3) suivante :

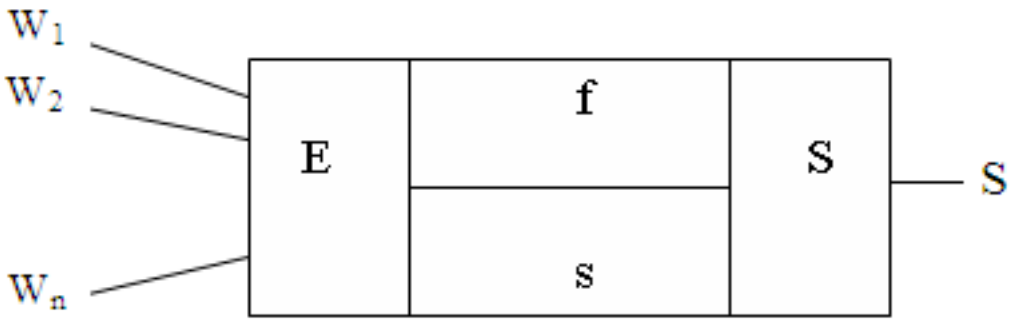

FIG. 1.3 – Le neurone formel.

Où W_i est le poids de l'entrée$_i$ (e_i), $E = h(e_1, ..., e_n)$ où h est la fonction d'entrée totale, f = fonction d'activation, s = seuil d'activation et S = réponse du neurone.

Le neurone, appelé aussi automate à seuil, calcule la somme pondérée (par les poids) de ses entrées. Il fournira une réponse positive si et seulement si cette somme dépasse un certain seuil, appelé seuil d'activation. Ce neurone partitionne en deux sous-ensembles l'ensemble des entrées, selon la parité de la réponse fournie. Le problème consiste donc à déterminer les entrées acceptables étant donné un jeu de poids définis. L'apprentissage par le biais d'exemples permettra d'ajuster ces coefficients (poids) par un algorithme de calcul de l'écart entre la sortie souhaitée et observée.

1.3 Gestion des connaissances

La technologie KDD se base sur un processus bien définit à plusieurs étapes [6] (voir FIG. 1.4) pour la découverte de connaissances d'un ensemble massif de données.

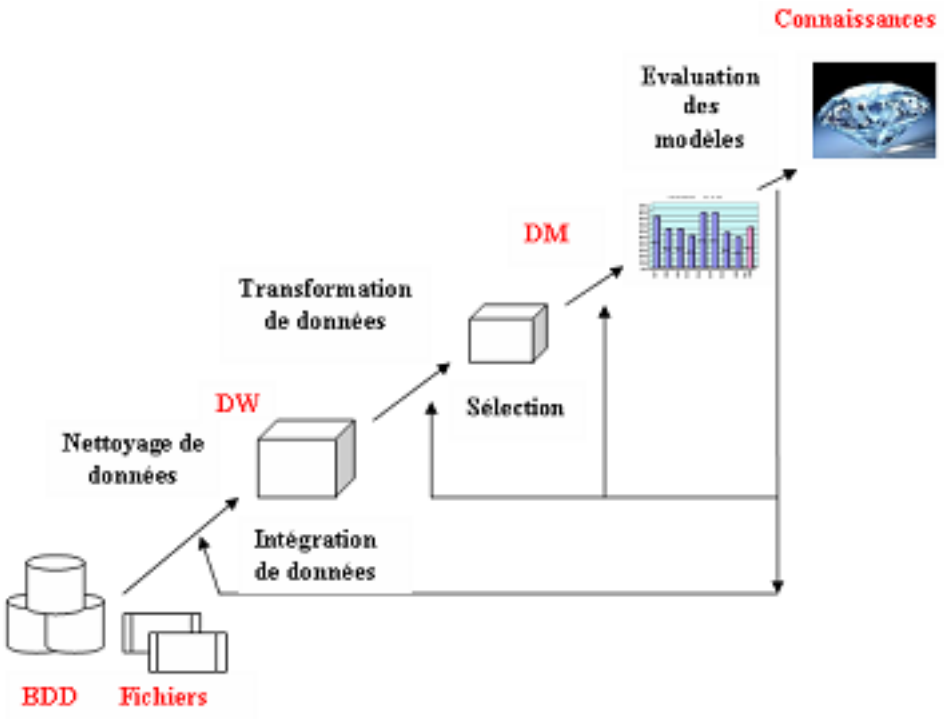

FIG. 1.4 – Le processus de découverte de connaissances KDD

1. *Nettoyage des données* : Pour éliminer le bruit et les données inconsistantes ;
2. *Intégration de données* : Où de multiples sources de données peuvent être combinées ;
3. *La sélection de données* : Dans cette étape se fait l'extraction de l'entrepôt de données pertinentes à la tâche d'analyse ;
4. *La transformation de données* : Les données sont transformées ou consolidées en des formes appropriées pour faciliter l'extraction et cela en effectuant par exemple des opérations d'agrégation ou de synthèse ;
5. *Extraction de données (DM)* : C'est un processus essentiel où sont appliquées des méthodes intelligentes afin d'extraire les modèles de données ;
6. *Évaluation d'un modèle* : Cette étape permet d'identifier les modèles intéressants qui représentent des connaissances ;
7. *Représentation de connaissances* : Utilisation des techniques de visualisation et de représentation pour présenter les connaissances extraites à l'utilisateurs.

Les étapes 1 à 4 sont des formes de pré-traitement de données où les données sont préparées au processus de DM.

Malgré l'énorme potentiel du processus KDD, les connaissances résultantes de son application sur les ensembles de données ne sont pas efficacement utilisées et ceci est dû aux contraintes suivantes :

- Les systèmes actuels de data mining exigent des utilisateurs de la technologie KDD qu'ils soient des experts car la compréhension claire et complète du processus KDD est essentielle pour réussir à avoir les connaissances qui nous intéressent ;
- Comme les connaissances résultantes du processus KDD sont en très grande quantité et aucun système ne s'occupe de leur organisation, les utilisateur préfèrent de refaire le processus KDD pour générer de nouvelles connaissances au lieu de réutiliser celles existantes ;
- Le processus KDD ne permet pas aux utilisateurs de spécifier leurs besoins en extraction d'une manière dynamique ; ils doivent les spécifier au début du processus KDD) ;
- Problème de gestion efficace de la connaissance dans l'extraction de données distribuée (DDM).

Tous les problèmes cités sont dus au manque d'organisation et de gestion des sorties des phases de processus KDD surtout celle de la dernière phase qui consiste en connaissances. D'où le besoin d'intégration des deux tâches extraction et gestion de connaissances. Pour remédier à ces problèmes, les chercheurs ont développé des techniques qui permettent de mieux gérer les connaissances découvertes afin de faciliter la tâche de recherche de connaissances aux utilisateurs. Dans ce qui suit, nous présentons quelques unes de ces techniques.

1.3.1 Framework d'extraction intensionnelle

Le framework d'extraction intensionnelle [12] a pour but de gérer l'extraction de connaissances par amélioration de processus d'extraction KDD et ceci se fait par spécification des besoins d'extraction des utilisateurs sous forme de schéma de découverte de connaissances (extraction intensionnelle). Avant de présenter ce framework, nous exposons d'abord le concept d'extraction intensionnelle.

1.3.1.1 L'extraction intensionnelle

L'extraction intensionnelle se divise en trois phases à savoir : la planification, l'accumulation et l'extraction.

1. *Planification :* Pendant cette phase, la spécification détaillée du processus KDD est stockée sous forme d'un schéma de découverte de connaissances (KDS) elle est réalisée en collaboration par l'analyste de data mining, l'expert du domaine traité et l'utilisateur final de connaissances recherchées. Le schéma KDS est compilé ; les

méta-données sont créées pour être utilisées au cours des deux dernières phases.

2. *Accumulation :* Au cours de cette phase, l'entrepôt de données est pré-traité et agrégé en consultation avec les méta-données pour rapporter les connaissances concentrées (KCs). Ces KCs stockent les formes intermédiaires de connaissances intensionnelles (selon le KDS) et servent de fenêtres de connaissances condensées pour les futures extractions.

3. *Extraction :* Cette phase est invoquée par l'utilisateur lorsqu'une requête d'extraction est présentée au système ou une application d'extraction est exécutée. Les KCs sont traitées par l'algorithme d'extraction pour découvrir les connaissances intensionnelles au cours de cette phase.

L'accumulation périodique des entrepôts de données à des intervalles réguliers donne lieu à une séquence de connaissances concentrées fournissant des fenêtres non chevauchées dans la BDD. Ces fenêtres sont à la base du renouvellement continu et de partage des connaissances car si plusieurs utilisateurs ont le même KDS alors ils peuvent exploiter les mêmes KCs et éviter ainsi le coût énorme de leurs générations.

1.3.1.2 Le modèle I-MIN

Le modèle I-MIN (Intension Mining Model) est un modèle centré utilisateur, dont les exigences d'extraction sont précisées sous forme d'un schéma de découverte de connaissances. Il est basé sur le concept d'extraction intensionnelle et permet la découverte et la gestion de connaissances dans les entrepôts de données.

Ce modèle réorganise le processus KDD en six étapes comme le montre la figure (FIG. 1.5).

- *L'étape IM1 :* C'est la première étape du processus KDD dans laquelle nous essayons de comprendre les données ensuite nous formalisons les besoins d'extraction. Elle correspond à la phase de *planification* dans l'extraction intensionnelle. Au cours de cette étape, les objectifs de découverte sont identifiés et les spécifications génériques du processus KDD sont stockées sous forme de schéma de découverte de connaissances. Le schéma est compilé et les méta-données résultantes sont stockées pour une utilisation future.

- *L'étape IM2 :* Cette étape correspond à la phase d'*accumulation*. Elle effectue le pré-traitement et l'agrégation sans l'intervention humaine car les fonctions néces-

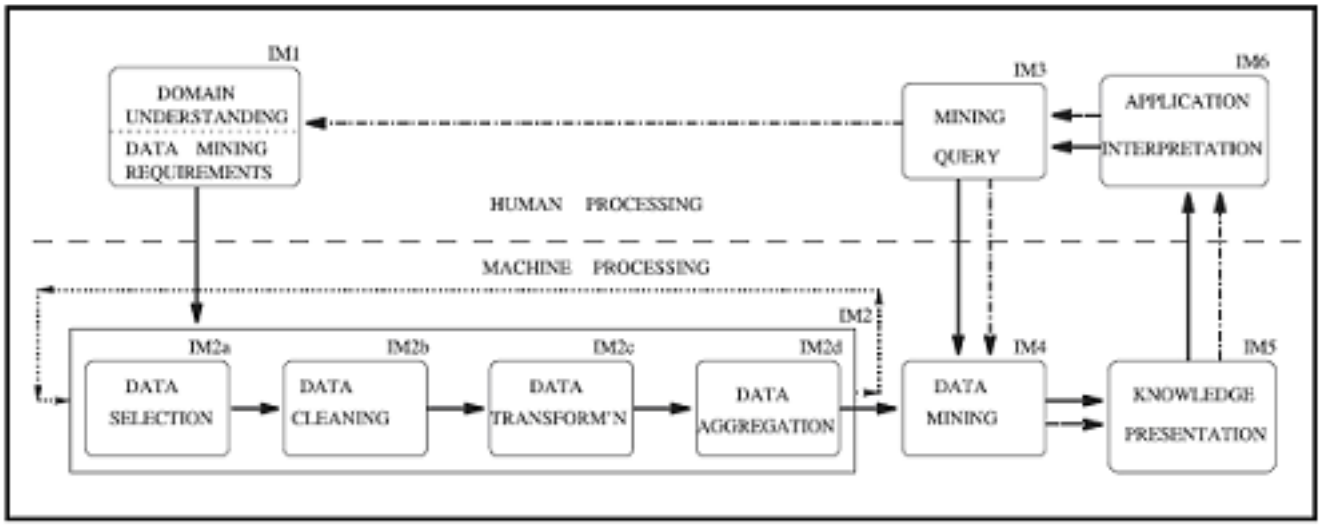

FIG. 1.5 – Le processus I-MIN de découverte de connaissances.

saires pour ces deux opérations sont spécifiées dans le KDS. L'étape IM2 est une étape composée de plusieurs sous-étapes dans laquelle IM2a-IM2c sont associées aux tâches de *sélection*, de *nettoyage* et de *transformation*, respectivement, dans le processus KDD traditionnel. L'analyse et l'agrégation des données pré-traitées qui sont effectuées lors de l'étape d'*extraction* du modèle sont effectuées à l'étape IM2d. Cette étape implique notamment l'exécution de la première phase de l'algorithme d'extraction. Le résultat de cette étape est une connaissance concentrée (KC).

– *L'étape IM3 :* Cette étape concerne le déclenchement de la phase d'extraction dans laquelle les utilisateurs finaux formulent leurs requêtes d'extractions, qui sont compilées conformément au schéma KDS généré dans l'étape IM1. Elle fait appel ensuite à l'algorithme d'extraction de données.

– *L'étape IM4 :* Au cours de cette étape, nous effectuons la fusion des KCs comme spécifié dans les requêtes ou applications puis la deuxième phase de l'algorithme d'extraction est invoquée et les connaissances attendues sont extraites à partir des KCs. Ces dernières peuvent être conservées et réutilisées par le développement d'applications pour répondre aux besoins complexes en connaissances.

– *Les étapes IM5 et IM6 :* Dans ces deux étapes, les connaissances résultantes sont représentées, interprétées et déployées respectivement comme dans le modèle de processus traditionnel KDD.

1.3.2 Le Data Mining distribué sur une grille

Les algorithmes de DM sont largement utilisés pour l'analyse de grand ensemble de données maintenus dans des noeuds géographiquement distribués afin d'utiliser la puissance de calcul des systèmes parallèles et distribués. Dans ce contexte, la grille peut jouer un rôle significatif en fournissant un support informatique effectif pour les applications distribuées de découverte de connaissance. Pour le développement des applications DM sur la grille Mario Cannataro et al 2004 [13] ont conçu un système appelé la grille de connaissance qui est présentée ci-dessous.

1.3.2.1 La grille de connaissances (K-Grid)

Le K-Grid est une architecture parallèle et distribuée qui intègre des techniques de DM et des technologies de grille dont l'objectif est d'appliquer le DM sur de grands ensembles de données répartis sur l'ensemble des noeuds d'une grille.

Cette architecture utilise les mécanismes de base de grille pour construire les services spécifiques de découverte de connaissances. Ces services peuvent être développés en utilisant les environnements de grilles disponibles. L'implémentation de ces services présentée dans [13] est basée sur le Globus Toolkit [14].

1.3.2.2 Les services de la grille de connaissances

La grille de connaissances se compose de deux couches comme le montre la figure (FIG. 1.6). La première est la couche noyau de K-Grid (Core K-Grid) et la seconde est la couche haut niveau de K-Grid (High level K-Grid).

Le *noyau de K-Grid* implémente les services de base pour la définition, la conception et l'exécution d'applications distribuées de découverte de connaissances sur la grille. Ses buts principaux sont la gestion de méta-données décrivant les caractéristiques des sources de données, des outils de DM, de la gestion de données et des algorithmes et outils de visualisation de données. De plus, cette couche coordonne l'exécution d'applications en essayant d'accomplir ses exigences sur les ressources disponibles d'une grille. Les deux services de cette couche sont :

– *Service de répertoire de connaissances (Knowledge Directory Service KDS) :* Ce service étend le service de base de surveillance et de découverte de Globus et contrôle les méta-données décrivant les données et les outils utilisés dans la grille de connaissances. Ceux-ci incluent les répertoires de données à extraire (sources de données), les outils et algorithmes utilisés pour l'extraction, la filtration et la manipulation

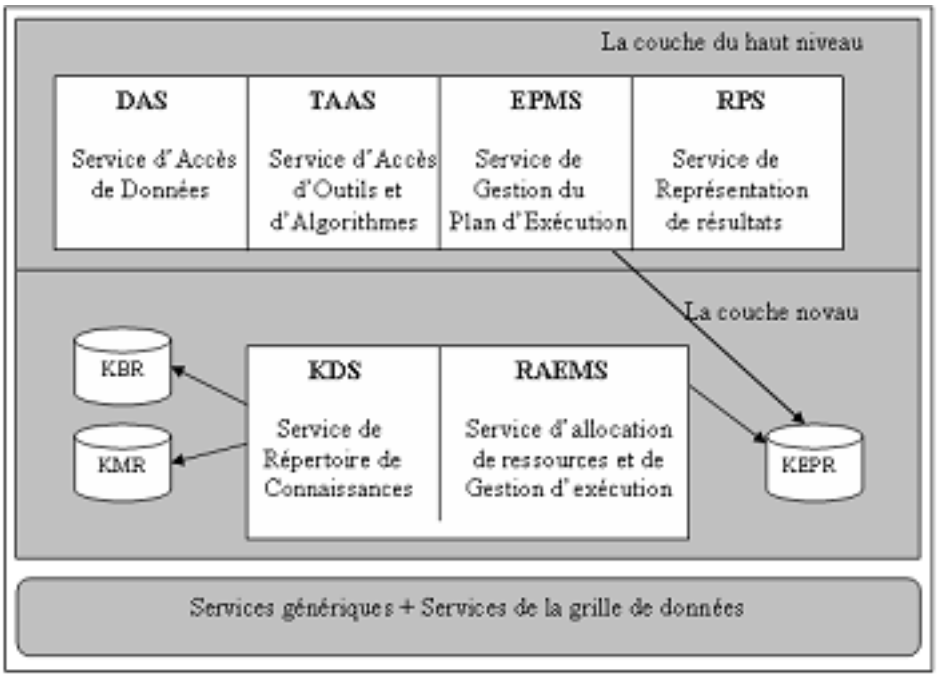

FIG. 1.6 – L'architecture de la grille de connaissances

des données ; les outils pour fouiller les données, visualiser et stocker les résultats de la fouille, les plans d'exécution distribués qui représentent une description abstraite de l'application distribuée de DM et les connaissances obtenues comme résultats du processus de DM (modèles découverts et ceux d'apprentissages).

– *Service d'allocation de ressources et de gestion d'exécution (Resource Allocation and Execution Management Service RAEMS) :* Ce service est utilisé pour trouver la meilleure correspondance entre un plan d'exécution et les ressources disponibles tout en satisfaisant les besoins de l'application en termes de puissance de calcul, de mémoire de stockage, de latence, de largeur de bande du réseau ainsi que des contraintes de la grille. Une fois le plan d'exécution est activé, ce service gère et coordonne l'exécution de l'application.

Les répertoires manipulés sont :

1. *Répertoire de méta-données de connaissances (Knowledge Metadata Repository KMR) :* Ce répertoire est représenté en document XML et contient toutes les méta-données concernant les ressources de la grille et les outils de DM.
2. *Répertoire de base de connaissances (Knowledge Base Repository KBR) :* Ce répertoire contient les connaissances découvertes par le processus de DM.
3. *Répertoire du plan d'exécution de connaissance (Knowledge Execution Plan Repository KEPR) :* Ce répertoire stocke les plans d'exécution des processus de DM.

Après avoir présenté la première couche de la grille de connaissance, nous passons à la deuxième couche qui est le *haut niveau de K-Grid*. Cette couche inclut les services utilisés pour composer, valider et exécuter un calcul parallèle et distribué de découverte de connaissances. D'autre part, cette couche offre des services de stockage et d'analyse de connaissances découvertes. Les services principaux sont :

- *Service d'accès de données (DAS) :* Ce service est responsable de la recherche, la sélection, l'extraction, la transformation et la livraison des données à traiter par les outils de DM. La recherche et la sélection sont basées sur le service KDS.

- *Service d'accès aux outils et algorithmes (TAAS) :* Ce service se charge de la recherche, la sélection et le téléchargement des outils et algorithmes de DM qui sont publiés par chaque noeud à l'aide du service KDS.

- *Service de gestion du plan d'exécution (EPMS) :* Un plan d'exécution est illustré par le graphe de la figure (FIG. 1.7) où les noeuds représentent les ressources et les arrêtes représentent l'interaction et les flux de données à travers les ressources. Le service EPMS prend les données et les programmes sélectionnés par l'utilisateur et génère un plan d'exécution abstrait décrivant le calcul conçu qui est ensuite mappé aux ressources concrètes de la grille.

- *Service de présentation des résultats (RPS) :* Ce service spécifie comment générer, présenter et visualiser les modèles de connaissances extraits (exemples : règles d'associations, modèles de clustering, modèles de classification) puis les stocker dans le KBR.

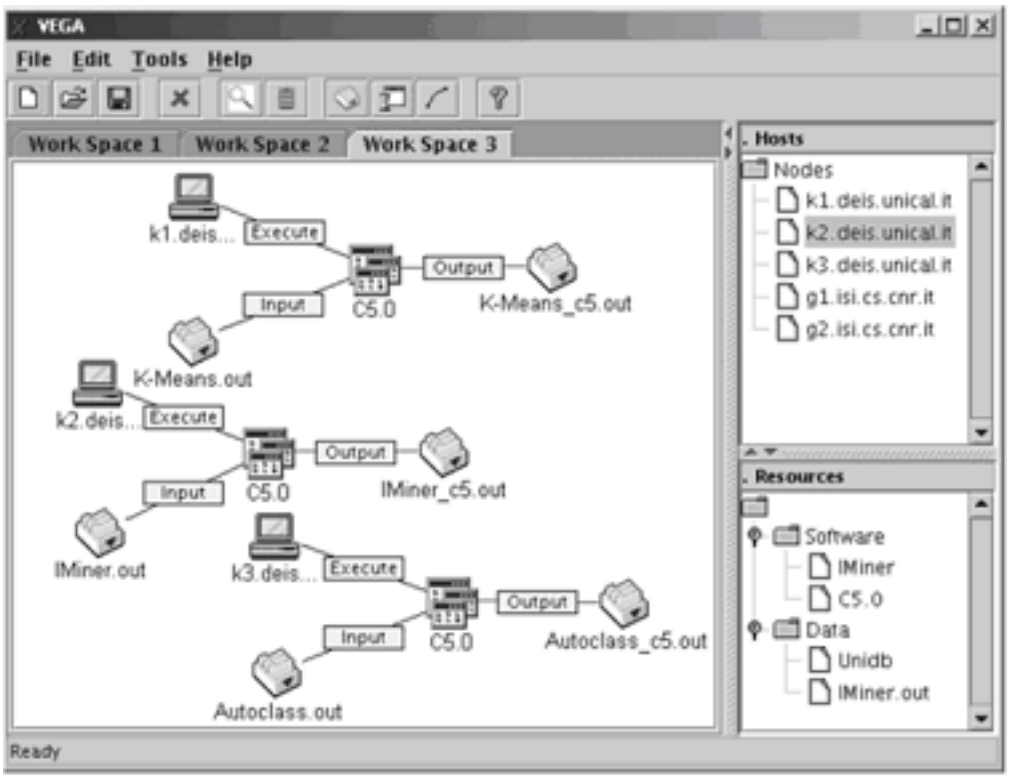

FIG. 1.7 – Exemple du plan d'exécution sous forme d'un graphe.

1.3.2.3 Conception d'une application DM de grille

Le processus de conception des applications distribuées de découverte de connaissance dans la grille de connaissance est montré dans la figure (FIG. 1.8). Il commence par la recherche et la sélection des ressources qui composent l'application, ceci est fait à l'aide des services DAS et TAAS qui analysent les méta-données (stockées dans leurs KMRs) représentant les ressources disponibles des noeuds de la grille. Les métas-données sur les ressources sélectionnées (noeuds de calcul, les sources de données et logiciels) pour le calcul sont ensuite stockées dans le répertoire de métas-données de tâche (TMR).

La conception de calcul DM est effectuée au moyen du service EPMS qui assure les opérations suivantes :

1. *Composition de la tâche :* Cette opération est réalisée par la définition des entités impliquées dans le calcul et les spécifications des relations entre elles ;
2. *Vérification de la consistance du calcul planifié :* Cette opération vérifie la consistance des liens entre les entités et détecte l'existence des erreurs (par exemple l'utilisation d'un noeud qui n'appartient pas à la grille) ;
3. *Génération du plan d'exécution :* Dans cette opération, le modèle de calcul est translaté en un plan d'exécution représenté par un document XML.

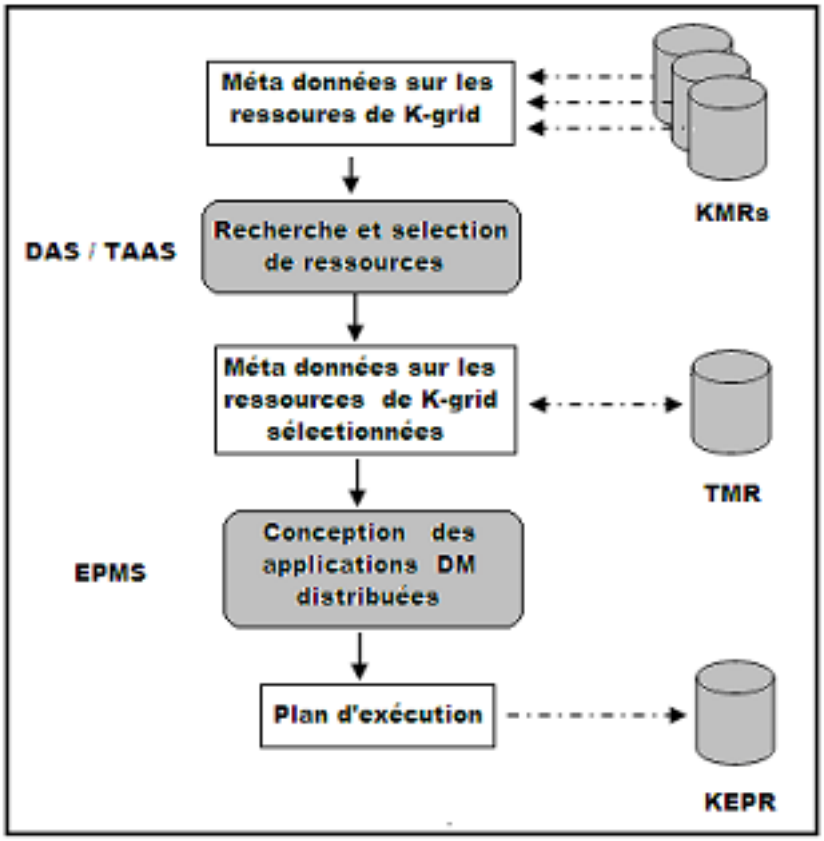

FIG. 1.8 – Conception de processus de calcul DM

1.3.3 La carte de connaissances pour les communautés virtuelles

La communauté virtuelle est définie par le rassemblement de personnes avec un intérêt commun pour partager les informations et coordonner leurs travaux via les technologies d'information comme les transactions [15].

La carte de connaissances est définie comme la classification des documents caractérisés par des concepts utilisés par les personnes appartenant à la communautés virtuelle qui s'intéressent aux documents [15].

Dans le but de faciliter la gestion de connaissances dans les communautés virtuelles et de raccourcir le cycle d'apprentissage, c'est-à-dire, un individu peut exploiter l'expérience des autres pour agrandir les siennes en partageant la connaissance, Fu-ren Lin et al 2006 [15] ont proposé la création d'une carte de connaissance en utilisant l'extraction d'information et des techniques de DM.

1.3.3.1 Création de carte de connaissances

La création d'une carte de connaissances dans une communauté virtuelle peut se faire soit par la classification ou par le clustering de documents qui sont tous les deux basés sur les mots clés des documents. Cependant, le clustering est plus efficace que la classification puisqu'il n'a pas besoin de déterminer à priori les classes par un expert. Les techniques d'extraction d'informations peuvent être utilisées pour transformer les documents non structurés en données structurées, et les techniques de DM peuvent être appliquées pour découvrir les relations entre les documents. Ces techniques incluent l'indexation de texte, l'extraction de mots clés, les méthodes Term Weighting et le clustering.

- *Indexation de texte :* Les documents non structurés sont d'abord indexés dans un arbre PAT-Tree [16] pour permettre l'accès efficace aux sources de données, où le nombre d'occurrence de mot est connu.

- *Extraction de mots clés :* Dans cette étape, le programme *auto_tag* est utilisé pour identifier les informations morphologiques de chaque mot dans des phrases, et écarter tous les mots sauf les noms composants les phrases. Puis, l'algorithme d'extraction de mots clés proposé par L.-f. Chien 1997 [16] est employé pour extraire les mots clés des expressions nominales continues. Après cette étape, le document non structuré est transformé en données structurées.

- *Term Weighting :* Le Term Weighting est une méthode statistique qui permet d'identifier les mots clés représentatifs pour chaque document permettant ainsi de le distin-

guer des autres documents. Nous notons que le modèle d'espace vectoriel est souvent utilisé pour la représentation de documents, où chaque document est représenté par un vecteur multidimensionnel dont chaque dimension correspond à un seul mot clé.

– *Clustering :* Les techniques de clustering sont utilisées pour spécifier les relations entre les documents en se basant sur leurs caractéristiques extraites. Dans cette étape, le clustering à deux étages (montré dans la figure (FIG. 1.9) est utilisé pour mettre en oeuvre les fonctions de création et de maintenance de la carte de connaissances. Premièrement, le clustering hiérarchique est appliqué (étage 1) pour obtenir la relation hiérarchique des documents. Puis, il faut déterminer le meilleur nombre de clusters en utilisant le *coefficient de silhouette (sc)* [17] (il faut avoir un compromis entre le raffinement des clusters et la complexité de l'algorithme hiérarchique) pour effectuer par la suite le partitionnement physique avec le clustering *k-mean* (étage 2).

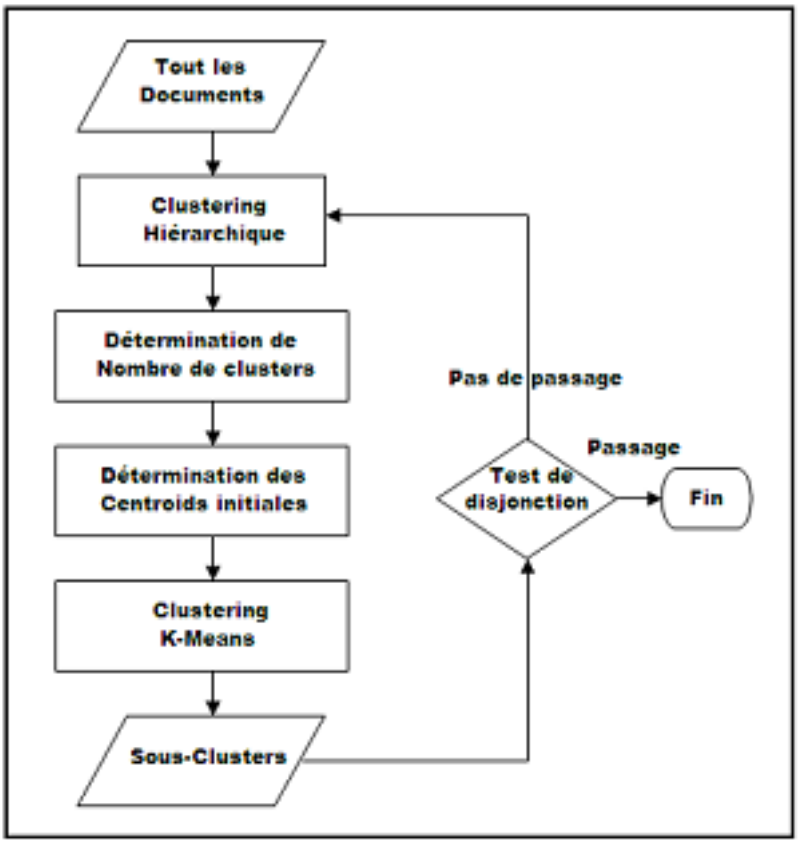

FIG. 1.9 – Processus de création de la carte de connaissance

1.3.3.2 Maintenance de la carte de connaissances

La carte de connaissances de la communauté virtuelle varie dynamiquement suite aux contributions continues des personnes, ce qui engendre l'ajout asynchrone d'objets de connaissances (documents) d'où la nécessité de sa maintenance pour le classement de nouveaux documents afin de la garder cohérente sans refaire son processus de clustering car dans le cas contraire cela engendre un coût informatique énorme. Les étapes essentielles de l'approche de maintenance sont illustrées dans la figure (FIG. 1.10).

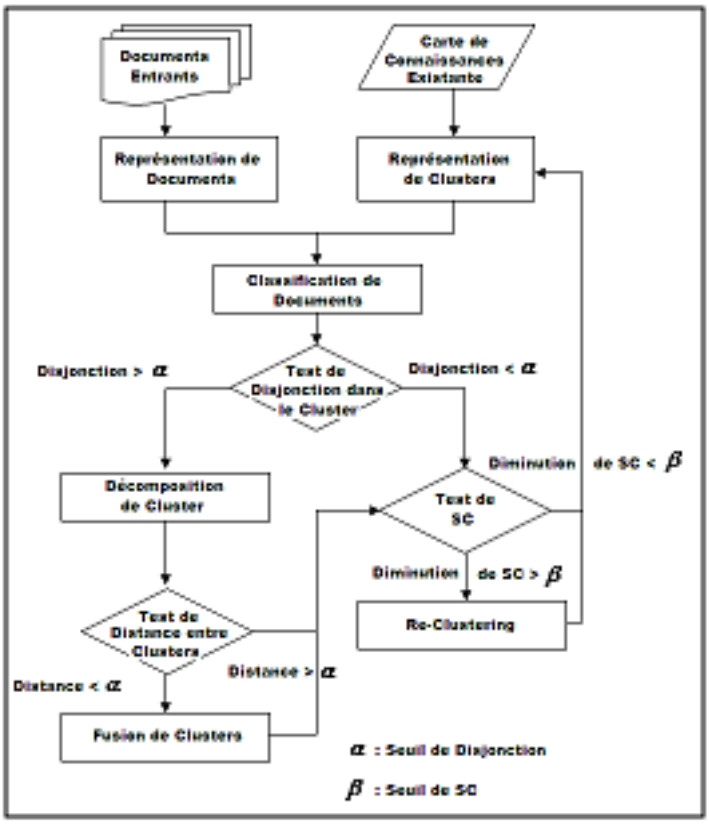

FIG. 1.10 – La procédure de maintenance de la carte de connaissance

– *Représentation de documents et de clusters :* Dans cette étape, les clusters résultants de la carte ainsi que les nouveaux documents sont représentés par des vecteurs de même dimension. Ainsi, nous pouvons comparer la *similarité* entre les clusters existants et les nouveaux documents en calculant la *distance euclidienne* entre le vecteur de document dont les valeurs représentent les occurrences des mots clés dans le document et le vecteur centroid de cluster dont les valeurs représentent la moyenne des occurrences de mots clés dans le cluster.

– *Classification de documents :* Nous calculons dans cette étape la *dissimilitude* entre le vecteur de nouveau document et les *vecteurs centroid* de clusters pour classer le

document dans le cluster avec lequel il a la plus petite *distance euclidienne*. Ensuite, un test de disjonction (qui est représenté par la *distance moyenne* entre les documents d'un même cluster) dans le cluster est effectué pour décider de décomposer le cluster résultant en sous clusters si le résultat du test dépasse un seuil α.

– *Décomposition et fusion de clusters :* Dans le cas où la distance est supérieure à α, le clustering à deux étages est appliqué pour décomposer le cluster en sous clusters dont le nombre est déterminé par le *coefficient silhouette (sc)*. Les sous clusters sont ensuite comparés aux clusters existants. Si la distance entre un sous cluster et un autre cluster existant est inférieure à α alors les deux sont fusionnés.

– *Re-clustering :* Dans cette étape, une comparaison est faite entre le nouveau *sc* et l'ancien et dans le cas d'un dépassement d'un seuil β, le nombre de sous clusters d'un super cluster donné n'est pas acceptable d'où la nécessité de refaire le clustering pour découvrir la nouvelle carte de connaissances dans le super cluster. Dans le cas où la différence entre l'ancien et le nouveau *sc* ne dépasse pas le seuil β alors la structure du cluster est toujours acceptable et aucun autre re-clustering n'est nécessaire.

1.4 Synthèse sur les techniques de gestion de connaissances

D'après la présentation des différents projets de recherche (en sections 1.3.1, 1.3.2 et 1.3.3) sur la gestion de connaissances dans le DM, nous remarquons que peu d'entres eux traitent le DDM. Si nous regardons le framework d'extraction intensionnelle, il englobe le processus KDD tout en assurant le partage de connaissances intermédiaires sous forme de connaissances concentrées (KCs) mais il ne gère pas les connaissances finales extraites dans ce processus. Pour la carte de connaissance pour la communauté virtuelle, elle aborde les problèmes de gestion de la connaissance. Cependant, elle propose des solutions pour le DM centralisé.

Le projet qui traite les problèmes de DDM à grande échelle est le K-Grid (la grille de connaissance) mais malheureusement, il fournit une manière simple de gérer la connaissance mais n'offre pas l'intégration et la coordination des résultats.

1.5 Les ontologies

L'utilisation de connaissances en informatique a pour but d'éviter la manipulation aveugle des informations par la machine et de permettre un dialogue et une coopération entre le système et les utilisateurs. Pour cela, le système doit avoir accès non seulement aux termes utilisés par l'être humain mais également à la sémantique qui leur est associée pour assurer une communication efficace. Les ontologies visent à représenter cette connaissance pour être à la fois interprétables par l'homme et par la machine. Ces dernières sont un sujet de recherche dans diverses communautés notamment l'ingénierie des connaissances, la recherche d'information et le traitement du langage naturel, les systèmes d'information coopératifs, l'intégration intelligente d'information et la gestion des connaissances. Les ontologies fournissent une connaissance partagée et commune sur un domaine qui peut être échangée entre des personnes et des systèmes hétérogènes.

1.5.1 Définition d'ontologies

Les ontologies permettent la représentation et la structuration des connaissances. En général, le terme ontologie employé dans des contextes variés tels que la linguistique, la philosophie ou l'intelligence artificielle, désigne une description de notions, ou concepts, et des diverses relations entre eux. Une ontologie peut alors être définie comme une *"formalisation explicite d'une conceptualisation partagée"* [18]. L'interprétation de cette définition a été donnée par Broekstra et al 2001 [19] :

- Conceptualisation réfère à un modèle abstrait de certains phénomènes dans le monde qui identifie les concepts appropriés de ce phénomène.
- Explicite signifie que les types de concepts utilisés et leurs contraintes sur leurs utilisation doivent être explicitement définis.
- Formalisation réfère au fait qu'une ontologie doit être compréhensible par la machine. C'est-à-dire cette dernière est capable d'interpréter la sémantique de l'information fournie.
- Partagée indique que l'ontologie supporte la connaissance consensuelle, et elle n'est pas restreinte à certains individus mais acceptées par un groupe.

Les connaissances traduites par une ontologie sont captées à l'aide des deux principaux éléments [20] à savoir : Concept et Relation. Nous détaillons ci-après ces deux éléments.

❖ Concepts

Les connaissances portent sur des objets auxquels nous nous référons à travers des concepts. Un concept peut représenter un objet matériel, une notion, une idée [21]. Il peut être divisé en trois parties : un terme (ou plusieurs), une notion et un ensemble d'objets. La notion, également appelée intension du concept, contient la sémantique du concept, exprimée en termes de propriétés et d'attributs, de règles et de contraintes. L'ensemble d'objets, également appelé extension du concept, regroupe les objets manipulés à travers le concept ; ces objets sont appelés instances du concept. Par exemple, le terme *"table"* renvoie à la fois à la notion de table comme objet de type *"meuble"* possédant un plateau et des pieds, et à l'ensemble des objets de ce type.

Les propriétés principales pouvant être associées à un concept sont les suivantes :

- *Généricité :* Un concept est générique si son extension est un ensemble vide ;
- *Identité :* Un concept porte une propriété d'identité si cette dernière permet de conclure quant à l'identité de deux instances de ce concept. Le concept d'étudiant porte une propriété d'identité liée au numéro de l'étudiant ;
- *Rigidité :* Un concept est rigide si toute instance de ce concept reste instance dans tous les mondes possibles. Exemple : humain est un concept rigide, étudiant est un concept non rigide ;
- *Anti-rigidité :* Un concept est anti-rigide si toute instance de ce concept est essentiellement définie par son appartenance à l'extension d'un autre concept. Exemple : étudiant est un concept anti-rigide car l'étudiant est un humain ;
- *Unité :* Un concept est un concept unité si, pour chacune de ses instances, les différentes parties de l'instance sont liées par une relation qui ne lie pas d'autres instances de concepts. Exemple : les deux parties d'un couteau, manche et lame sont liées par une relation *"emmanché"* qui ne lie que la lame et son manche.

Les propriétés portant sur deux concepts sont :

- *Equivalence :* Deux concepts sont équivalents s'ils ont la même extension.
- *Disjonction (incompatibilité) :* Deux concepts sont disjoints si leurs extensions sont disjointes. Exemple : homme et femme ;
- *Dépendance :* Un concept C1 est dépendant d'un concept C2 si pour toute instance de C1 il existe une instance de C2 qui ne soit ni partie ni constituant de l'instance de C2. Exemple : parent est un concept dépendant de enfant (et vice-versa).

❖ Relations

Les relations représentent des interactions entre concepts permettant de construire des représentations complexes de la connaissance du domaine [22]. Elles établissent des liens

sémantiques binaires, organisables hiérarchiquement. Exemple : les concepts *"Personnalité"* et *"Film"* sont reliés entre eux par la relation sémantique *"réalise (Personnalité, Film)"* dans laquelle *"Personnalité"* est le domaine et *"Film"* est le rang.

Les propriétés intrinsèques à une relation sont :

- *Propriétés algébriques :* Symétrie, réflexivité et transitivité ;
- *Cardinalité :* La cardinalité représente le nombre possible de relations de ce type entre les mêmes concepts (ou instances de concept). Les relations portant une cardinalité représentent souvent des attributs. Exemple : une pièce a au moins une porte.

Les propriétés liant deux relations sont :

- *Incompatibilité :* Deux relations sont incompatibles si elles ne peuvent lier les mêmes instances de concepts. Exemple : les relations *"être rouge"* et *"être vert"* sont incompatibles ;
- *Inverse :* Deux relations binaires sont inverses l'une de l'autre si, quand l'une lie deux instances I1 et I2, l'autre lie I2 et I1. Exemple : les relations *"a pour père"* et *"a pour enfant"* sont inverses l'une de l'autre ;
- *Exclusivité :* Deux relations sont exclusives si, quand l'une lie des instances de concepts, l'autre ne lie pas ces instances, et vice-versa. L'exclusivité entraîne l'incompatibilité. Exemple : l'appartenance et la non appartenance sont exclusives.

Les propriétés liant une relation et des concepts sont :

- *Lien relationnel :* Il existe un lien relationnel entre une relation R et deux concepts C1 et C2 si, pour tout couple d'instances des concepts C1 et C2, il existe une relation de type R qui lie les deux instances de C1 et C2. Un lien relationnel peut en outre être contraint par une propriété de cardinalité, ou porter directement sur une instance de concept [23]. Exemple : il existe un lien relationnel entre les concepts *"texte"* et *"auteur"* d'une part et la relation *"a pour auteur"* d'autre part ;
- *Restriction de relation :* Pour tout concept de type C1, et toute relation de type R liant C1, les autres concepts liés par la relation sont d'un type imposé. Exemple : si la relation *"mange"* portant sur une *"personne"* et un *"aliment"* lie une instance de *"végétarien"*, concept subsumé par *"personne"*, l'instance de *"aliment"* est forcément instance de *"végétaux"*.

1.5.2 Typologies d'ontologies

La construction d'une ontologie doit se faire à partir d'un champ de connaissances bien délimité par un objectif opérationnel clair, et portant sur des connaissances objec-

tives dont la sémantique puisse être exprimée rigoureusement et formellement. Partant de là, plusieurs types d'ontologies peuvent être distinguées (voir FIG. 1.11) en fonction des différents objectifs opérationnels recensés [24].

❖ **Ontologie de représentation de connaissances**

Elle exprime des conceptualisations valables dans différents domaines. Elle décrit des concepts très généraux comme l'espace, le temps, les objets, les événements, les actions, etc. Ces concepts ne dépendent pas d'un problème ou d'un domaine particulier, et doivent être, du moins en théorie, consensuels à de grandes communautés d'utilisateurs. Son sujet est l'étude des catégories des choses qui existent dans le monde [25] [26] Dolce [27] et Sumo [28] sont des exemples de ce type d'ontologies.

❖ **Ontologie du Domaine**

Cette ontologie exprime des conceptualisations spécifiques à un domaine, elle est réutilisable pour plusieurs applications de ce domaine. Elle fournit les concepts et les relations permettant de couvrir les vocabulaires, activités et théories de ce domaine. Selon Mizoguchi 2000 [29], l'ontologie du domaine caractérise la connaissance du domaine où la tâche est réalisée. Par exemple, dans le contexte du e-learning, le domaine peut être celui de la formation. De nombreuses ontologies de domaine existent déjà, telles que MENELAS dans le domaine médical [30], ENGMATH pour les mathématiques [31], TOVE dans le domaine de la gestion des entreprises [32] etc.

❖ **Ontologie de Tâche (raisonnement)**

L'ontologie de tâche fournit un vocabulaire systématisé des termes employés pour résoudre des problèmes liés aux tâches (par exemple : enseigner, diagnostiquer, ...), qui peuvent être ou non du même domaine. Elle fournit un ensemble de termes au moyen desquelles nous pouvons décrire généralement comment résoudre un type de problèmes. Elle inclut des noms, des verbes et des adjectifs génériques dans les descriptions de tâches.

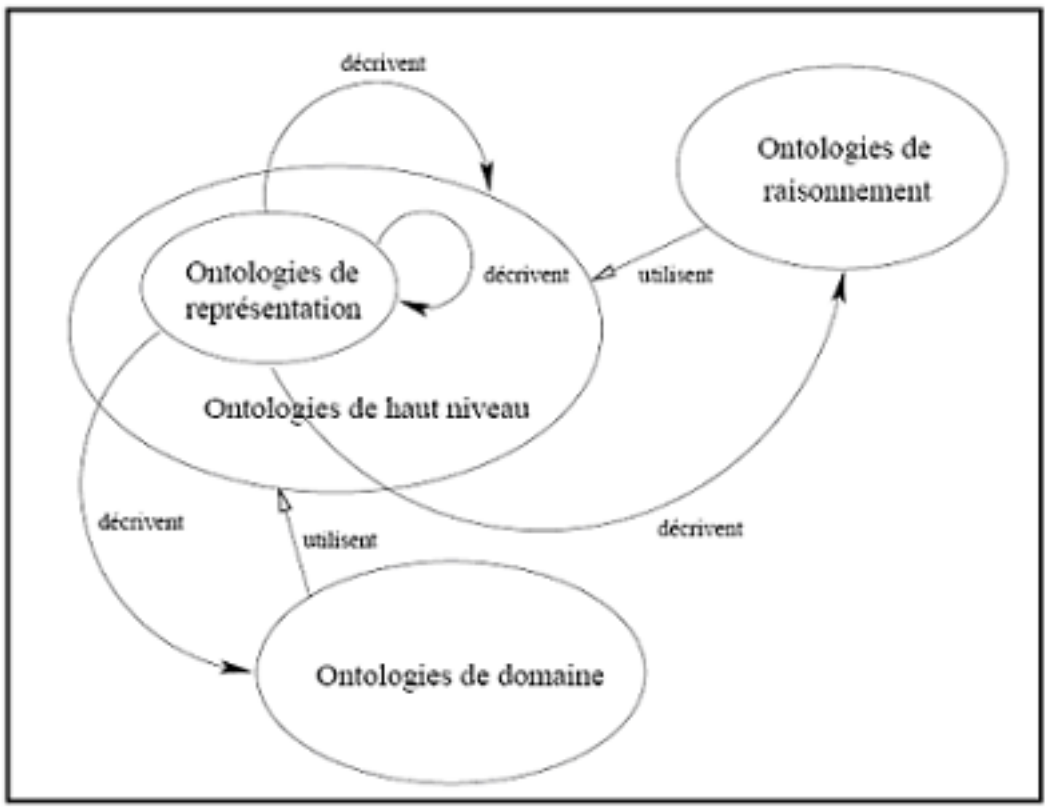

FIG. 1.11 – Les différents types d'ontologies

1.5.3 Langages de définition d'ontologies

Les ontologies sont apparues pour la première fois dans le cadre de web sémantique puis elles ont été intégrées dans plusieurs autres domaines. Les langages de définition d'ontologies [33] se résument par quelques couches (les couches 2, 3 et 4) de la pile de web sémantique proposée par Tim Berners-Lee et al 2001 [34] comme illustrée dans la figure (FIG. 1.12).

XML : XML est un standard d'échange de données sur le Web permettant de structurer les données mais sans communiquer leur sémantique. C'est un langage pour les données semi structurées. Il a été proposé comme solution pour des problèmes d'intégration de données, parce qu'il permet un codage et un affichage flexibles des données en employant des méta-données pour décrire la structure des données. Cependant, du point de vue de l'interopérabilité sémantique, XML a des limitations [35].

RDF : Au dessus de XML, le W3C a développé le langage de Ressource Description Framework (RDF) pour normaliser la définition et l'utilisation des méta-données. RDF emploie XML et il est à la base du Web sémantique tel que tous les autres langages correspondant aux couches supérieures sont établis sur lui. RDF est un langage simple de

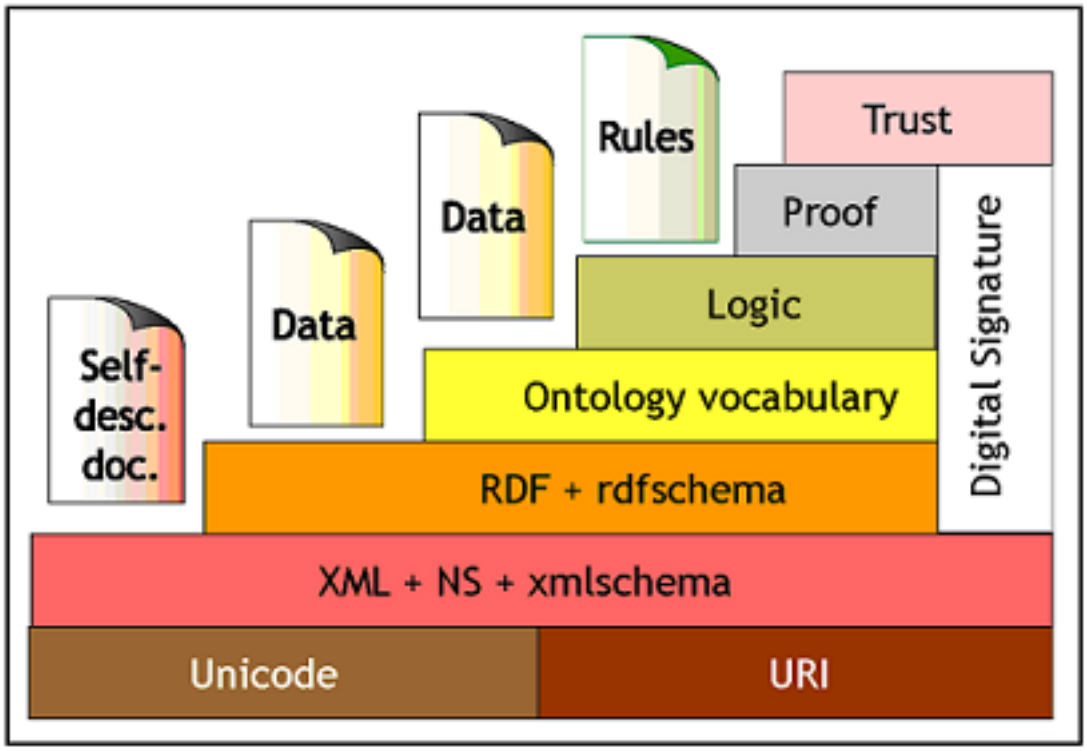

FIG. 1.12 – La pile de web sémantique

représentation d'information en Web et fournit un modèle pour décrire et créer des relations entre les ressources. RDF définit une ressource en tant qu'objet qui est uniquement identifiable par un URI. Les ressources ont des propriétés qui lui sont liées et qui sont identifiées par des propriétés-types qui ont des valeurs correspondantes. Les Propriété-types expriment les rapports des valeurs liées aux ressources. La structure de base de RD emploie fondamentalement des triples de RDF sous forme de (sujet, prédicat ou attribut, objet).

RDF Schéma : RDF schéma permet aux utilisateurs de définir des ressources (rdfs : Ressource) avec des classes, des propriétés et des valeurs. Le concept de classe dans RDFS (rdfs : La classe) est semblable au concept de la classe dans les langages de programmation orienté objet. Ceci permet à des ressources d'être définies comme des instances de classes. Une propriété de RDFS (rdfiProperty) peut être vue comme un attribut d'une classe. Les propriétés de RDFS peuvent hériter d'autres propriétés (rdfs :subPropertyOf), et les contraintes de domaine (rdfs :domain) et de rang (rdfs : range) peuvent être appliquées pour focaliser leur utilisation. Par exemple, une contrainte de domaine est employée pour limiter les classes qui peuvent avoir une propriété spécifique et une contrainte de rang est utilisée pour limiter ses valeurs possibles.

Ontologies : Précédemment, nous avons souligné que RDF et RDFS étaient les modèles de base et la syntaxe pour le Web sémantique. Au dessus de la couche RDF/S, il

est possible de définir des langages plus puissants pour décrire la sémantique. Le langage de balisage le plus en avant pour éditer et partager des données en utilisant les ontologies sur Internet est le langage d'Ontologie de Web (OWL). OWL ajoute une couche expressive puissante à RDF/S, fournissant des mécanismes puissants pour définir les structures conceptuelles complexes et décrire formellement la sémantique des classes et des propriétés utilisées dans des ressources de Web utilisant généralement le formalisme logique connu sous le nom de logique de description (DL).

1.5.4 Les outils de développement des ontologies

Plusieurs framework supportant les ontologies OWL sont disponibles. Nous présentons brièvement ceux qui sont les plus utilisés par la communauté [33], à savoir le framework Jena et l'API Protégé-OWL qui sont tous les deux disponibles pour le langage Java et ils sont open source ; ce qui est très intéressant car cela offre la possibilité de s'adapter aux besoins sémantiques futur des applications Web en intégrant facilement les composants sémantiques.

Jena : Jena (Jena 2002 ; Jena 2005) est un framework Java pour la construction des applications de Web sémantique développé par le programme de Web sémantique de laboratoire HP. Il fournit un environnement de programmation pour RDF, RDFS et OWL, incluant un moteur d'inférence basé sur les règles et un langage d'interrogation pour RDF appelé RDQL (2005). Notre intérêt étant porté sur les ontologies, nous discutons l'API Jena 2 Ontology incluse dans Jena toolkit. Cet API supporte plusieurs langages de description d'ontologies tels que DAML, DAML+OIL et OWL.

A noter que l'API Jena OWL supporte les trois sous langages de OWL, à savoir OWL Lite, OWL DL et OWL Full. Spécifiant un URI à une ontologie OWL, Jena analyse l'ontologie et crée un modèle. Avec ce modèle, il est possible de manipuler l'ontologie, de créer de nouvelles classes OWL, propriétés ou instances.

Comme nous l'avons souligné, Jena offre la possibilité de raisonnement et fournie trois raisonneurs différents qui peuvent être attachés à un modèle d'ontologie où chacun fournie un degré différent de raisonnement. L'aspect intéressant de Jena est que son moteur d'inférence est écrit d'une manière très générique de sorte qu'il permet aux développeurs d'écrire leurs propres règles d'inférence pour mieux adresser leurs besoins. Cette implémentation générique permet également d'attacher n'importe quel raisonneur qui est conforme avec l'interface DIG, qui est un standard fournissant un accès aux raisonneurs Racer, EaCT, et Pellet.

L'API Protégé-OWL : Protégé (Protégé 2005) est une plate forme libre, open source

qui fournit à la communauté d'utilisateur une suite d'outils pour construire les applications basées sur les connaissances avec des ontologies. Elle a été développée par les laboratoires médicaux d'informatique de Stanford de l'École de Médecine de Stanford. L'API Protégé-OWL est une bibliothèque de Java open source pour OWL et RDF(S). Elle fournit des classes et des méthodes pour charger et stocker les fichiers OWL pour interroger et manipuler les modèles de données OWL et pour raisonner (Protégé-API 2006). Cette API qui fait partie de plug-in de Protegé-OWL, étend le noyau de système de Protégé basé sur les frames de sorte qu'il puisse supporter les ontologies OWL et permettre aux utilisateurs de développer des plug-in de OWL pour le Protégé ou même de créer des applications autonomes. L'API Protégé-OWL emploie le framework Jena pour l'analyse et le raisonnement sur les ontologies OWL et fournit des supports additionnels pour programmer les interfaces graphiques utilisateurs basées sur la bibliothèque Java Swing.

1.6 Conclusion

Dans le présent chapitre, nous avons présenté un aperçu sur la représentation et l'organisation de connaissances dans différents domaines y compris celui de Data Mining. Il est à noter que les techniques d'organisation de connaissances vues dans la troisième section ne répondent pas suffisamment aux besoins de gestion de connaissances dans de grands systèmes distribués, le domaine qui nous intéresse dans ce mémoire. La cinquième section qui porte sur les ontologies nous montre que ces dernières ont des outils qui permettent de bien présenter les connaissances et de prendre en considération la relation entre eux et leur aspect sémantique, il est donc intéressant de les exploiter dans les techniques d'organisation de connaissances.

Chapitre 2

L'Architecture Knowledge Map (KM)

2.1 Introduction

Les techniques de data mining distribuées (DDM) existantes effectuent une analyse partielle des données locales sur différents noeuds puis génèrent des modèles globaux par agrégation des résultats locaux. Ces deux étapes ne sont pas indépendantes puisque les approches naïves d'analyse locale peuvent produire des modèles de données globaux qui sont incorrects ou ambigus.

Pour palier à ce problème, Le-Khac et al 2007 [36] présentent un outil appelé Knowledge Map (KM) pour représenter facilement et efficacement la connaissance extraite dans une plateforme grille distribuée à grande échelle. Le KM permet également de faciliter l'intégration/coordination des processus d'extraction locaux et de connaissances existantes pour augmenter l'exactitude des modèles finaux.

Ce deuxième chapitre est structuré en trois sections dont la première présente l'architecture de KM, la deuxième expose les opérations qui peuvent être appliquées sur cette dernière et la dernière section donne une évaluation de performances de cette architecture.

2.2 L'architecture de KM

L'architecture de KM est illustrée dans la figure (FIG. 2.1). Elle possède les composants suivants : le KM local, le noyau de KM, le navigateur de KM, le retriever de KM et le gestionnaire de KM.

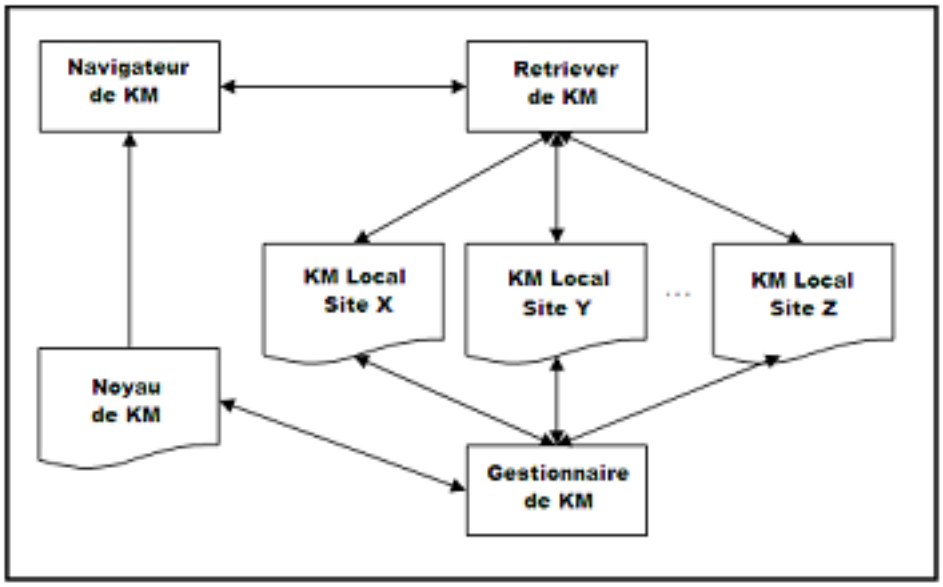

FIG. 2.1 – Architecture de KM

KM Local : Ce composant (FIG. 2.2) [36] se trouve dans chaque noeud local du système et représente un répertoire d'entrées. Chaque entrée représente une connaissance et possède deux parties : la méta-connaissance, qui inclut les propriétés et la description de la connaissance et le représentatif des entrées de la connaissance qui dépend du problème traité. Nous avons, par exemple, le représentatif de tâches de clustering et celui des connaissances à base de règles.

Pour les connaissances à base de règles (FIG. 2.2(b)), les connaissances extraites sont représentées comme des règles de production qui sont sous forme *"IF {prémisse} THEN {conclusion}"* avec la *prémisse* et la *conclusion* qui contiennent un ensemble d'items. Afin de représenter ces règles par leurs items, le représentatif dans cette approche comporte deux parties : la table de règle et la table d'index d'items. La première est la table de règles dans laquelle chaque ligne représente une règle incluant son identité, son contenu, ses attributs ainsi que ses informations de création. La table d'index d'items est une structure de donnée qui fait la correspondance des items à la table de règles tel que dans chaque ligne nous trouvons l'item et la liste des règles qui le contiennent.

Dans le cas de clustering (FIG. 2.2(a)), le représentatif des connaissances extraites se compose d'un seul ou plusieurs clusters. Un cluster a un ou plusieurs éléments représentatifs dont les champs sont remplis par l'utilisateur. La méta-donnée de ces champs est aussi incluse dans chaque représentatif. Le KM permet à l'utilisateur de définir cette

méta-donnée. En outre, le cluster contient des informations sur sa création pour montrer si le résultat est obtenu par clustering ou par un processus d'intégration qui fusionne des sous-clusters de différentes sources.

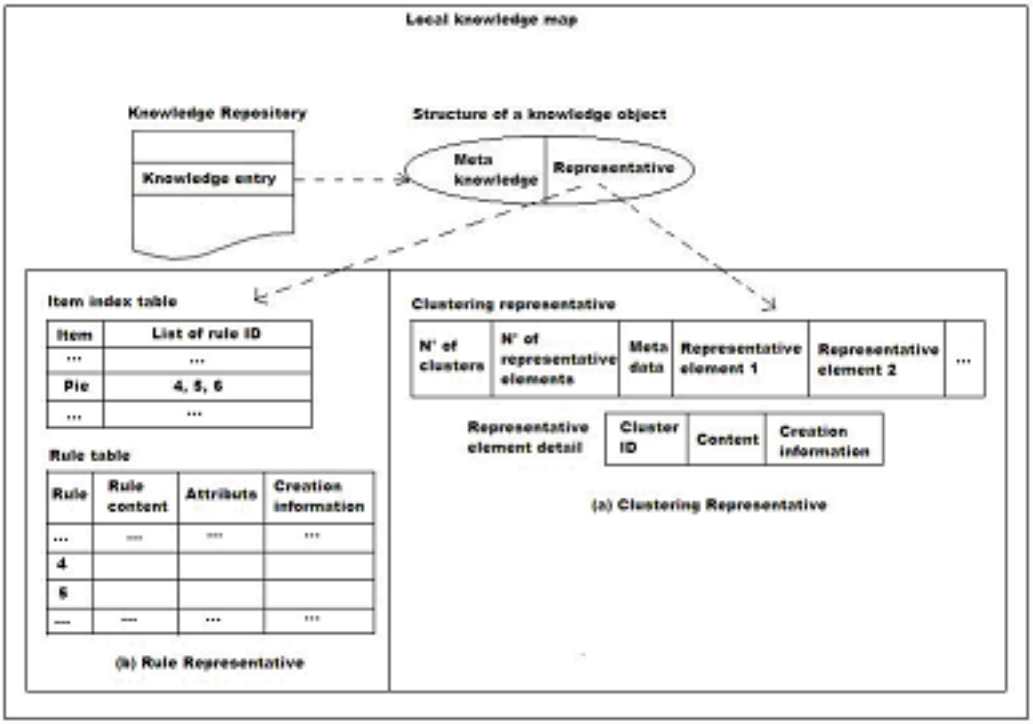

FIG. 2.2 – KM Local.

Noyau de KM : Ce composant (FIG. 2.3) [36] possède deux parties essentielles : la première représente le répertoire d'arbres de concepts pour le stockage d'un ensemble de domaines applicatifs (un arbre de concepts par domaine d'application) qui est définit par l'administrateur avant l'utilisation de KM et qui peut être mis-à-jour durant l'exécution. Dans cette approche, la connaissance extraite est assignée par l'utilisateur à un seul sous-domaine d'application. Comme illustré par la figure (FIG. 2.3), le répertoire d'arbre de concepts contient le domaine d'application *"meteorology"* qui contient les sous-domaines d'application *"Weather forecasting"*, *"Storm"*, *"Climat"* etc. La seconde partie de noyau de KM est le répertoire de méta-connaissances qui contient les méta-données de connaissances extraites de différents noeuds. Les méta-données de chaque connaissance sont représentées par une entrée de méta-connaissances. La figure (FIG. 2.3) montre un exemple d'entrée de méta-connaaissances qui inclut les méta-données de la connaissance extraite comme sa location (*Site = pcrgcluster.ucd.ie, Directory = /user/test*), l'identité de la connaissance (*ConceptID = 1122, "tropical cyclone"* dans ce cas), la tâche de DM (*"Clustering"*) etc. En se basant sur ces informations, l'utilisateur pourra déterminer les connaissances extraites qui l'intéressent.

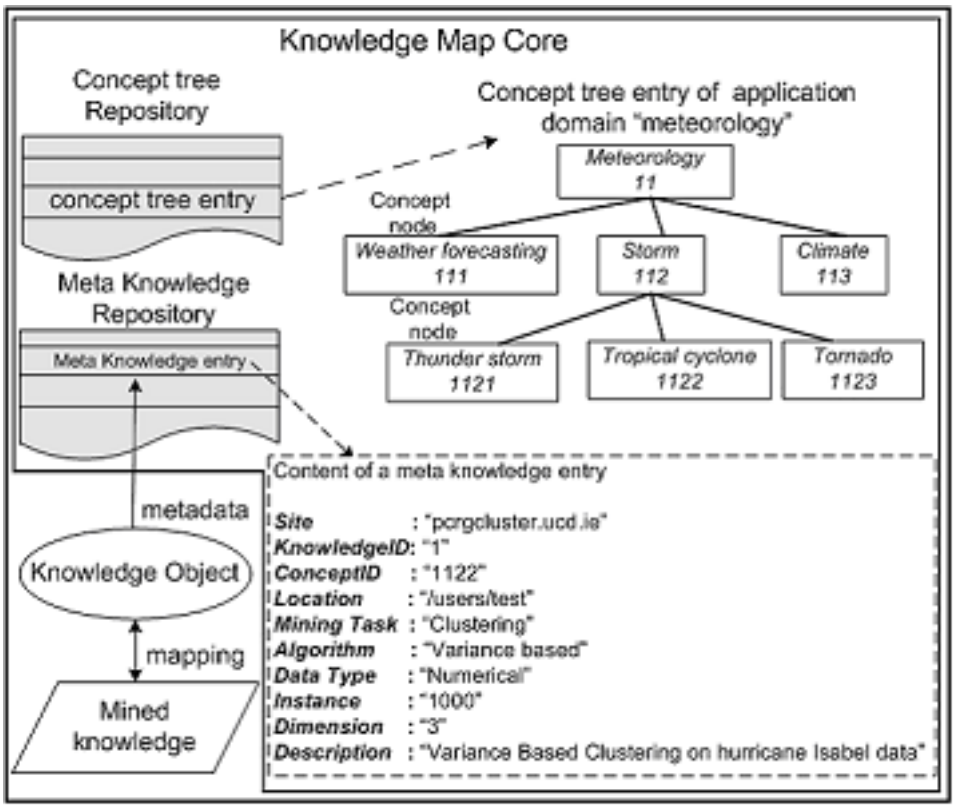

FIG. 2.3 – Noyau de KM.

Navigateur de KM : Ce composant aide les utilisateurs à raffiner leurs requêtes pour mieux explorer le KM et déterminer les connaissances qui les intéressent. Les résultats de cette tâche sont des méta-connaissances [36]. Par exemple, si l'utilisateur veut extraire quelques connaissances sur *"tropical cyclone"*. Le domaine d'application *"Meteorology"* est utilisé par ce composant pour permettre à l'utilisateur de parcourir le domaine *"tropical cyclone"* puis extraire la liste d'informations qui y sont reliées. En se basant sur ces méta-connaissances et leur domaine d'application, les utilisateurs décident quelles connaissances à extraire.

KM Retriever : Le rôle de ce composant est de rechercher les connaissances nécessaires en utilisant les informations fournies par l'utilisateur après l'exploration du domaine d'application et obtention des méta-connaissances nécessaires. Il interagit avec les noeuds locaux concernés et collecte les connaissances [36].

Gestionnaire de KM : Ce composant est responsable de la gestion/coordination de KM Local et de noyau de KM afin de garder la cohérence entre les deux. Pour le premier, le gestionnaire de KM fournit des primitives pour créer, ajouter, supprimer et mettre à jour les entrées de connaissances. Il permet également de suggérer les méta-connaissances locales au répertoire de méta-connaissances du noyau de KM. Ce composant fournit également des primitives pour manipuler les méta-connaissances dans le répertoire aussi bien

que les noeuds de concepts dans le répertoire d'arbres de concepts.

2.3 Opérations sur le KM

Dans cette section, nous présentons les opérations de base du système KM comprenant la création, la recherche et l'extraction de connaissances ainsi que la maintenance.

✍ **La création**

KM local : Premièrement, les éléments de connaissances de la même connaissance extraite sont structurés dans sa table de connaissances. Puis, la table d'index de chaque connaissance est constituée en scannant sa table de connaissance et son entête de connaissance est construite en se basant sur ces tables. Les informations de ces entêtes vont être envoyées au noyau de KM pour créer ou mettre à jour les répertoires de méta-connaissances.

Le noyau de KM : Premièrement, quelques domaines d'applications connus sont prédéfinis et leur hiérarchie de noeuds est établie mais à tout moment l'utilisateur peut ajouter de nouveaux domaines d'applications. A chaque noeud dans la hiérarchie est créée la liste de méta-connaissances en se basant sur les entêtes de connaissances des KM locaux.

✍ **Recherche et extraction de connaissances :** Le processus de recherche commence par la navigation de l'utilisateur pour explorer les domaines d'application puis il détermine un ensemble de noeuds des hiérarchies de domaines d'application à partir desquels il veut rechercher les connaissances. L'intersection des informations sur les connaissances est exécutée pour résumer les informations des connaissances reliées à tous les sous-domaines des noeuds des hiérarchies concernés. L'utilisateur choisit ensuite quelques connaissances en se basant sur leurs informations (location, propriétés et descriptions) pour rechercher leurs contenus en détail. Par la suite, l'utilisateur introduit les mots clés des termes relatifs contenus dans les connaissances. Le système KM utilise leur (Site ID, KnowledgeID) et les mots clés pour rechercher les éléments de connaissances des noeuds locaux concernés. Dans chaque noeud local, les entêtes relatives aux connaissances sont examinées et leur table d'index est utilisée pour déterminer les éléments de connaissances adéquats aux mots clés fournis. En fin, l'ensemble d'éléments satisfaisant les besoins sont rassemblés et envoyés au noeud requête.

✍ **La maintenance :** Cette fonction est conçue pour garder le système KM à jour en assurant les fonctionnalités suivantes : l'indexation des connaissances entrantes, la mise-à-jour des informations relatives aux connaissances existantes et la garantie de la cohérence entre les informations des connaissances entre le KM local et le noyau de KM.

2.4 Evaluation de performance du KM

La distribution de l'architecture KM est déployée sur la topologie TreeP (voir section 3 dans le chapitre 3) dont laquelle les noeuds physiques contiennent les KM locaux et les noeuds virtuels (qui sont aussi appelés serveurs) contiennent le noyau de KM qui est constitué de répertoires de méta-connaissances. Chaque noeud serveur manipule les méta-connaissances d'un groupe de noeuds physiques qui sont sous sa responsabilité (dans son sous arbre) [1].

L'une des tâches les plus importantes dans le Data Mining Distribué est le processus de fusion de modèles locaux pour construire les modèles globaux et la distribution faite selon le TreeP facilite l'atteinte de cet objectif : les modèles locaux aux noeuds feuilles (niveau 0) sont fusionnés à leur noeud serveur au niveau 1. Puis, les modèles au niveau 1 vont être hiérarchiquement fusionnés aux noeuds serveurs au niveau $2, 3, ..., h-1$ jusqu'au noeud racine. Comme résultat, les connaissances intégrées vont être stockées aux noeuds serveurs dans le KM et le noeud racine contient les modèles globaux de tout le système [37]. L'architecture KM est appliquée par ses concepteurs Le-Khac et al 2008 [1] [37] sur les résultats de techniques de clustering basées sur la variance [38]. La plate forme utilisée pour l'expérience consiste de 6 noeuds serveurs qui représentent un cluster hétérogène dont les noeuds varient entre (Intel PIII 720Mhz, 384Mb RAMg) à (Intel PIV 2Ghz, 512Mb RAMg) avec le noyau Linux 2.6.x. Le middleware utilisé pour la communication est Java RMI. Les répertoires de méta-connaissances sur chaque noeud serveur stocke 10000 à 20000 entrées tout dépend de leurs nombres de fils (chaque fils possède 5000 entrées de connaissances). Les répertoires de noyau de KM ainsi que ceux de KM local sont exprimés en format XML. Le temps d'exécution de KM inclut un ensemble de fonctions (*daemons*) *KM* (chaque noeud serveur possède un *daemon KM* qui est responsable des requêtes locales ainsi que celles distantes). L'application KM peut envoyer des requêtes à un ou plusieurs noeuds à travers les *daemons*. Les concepteurs de KM [37] ont exécutés les opérations de recherche d'un noeud de niveau 0. Ils ont effectué des tests avec 6, 12, 18 et 24 noeuds physiques selon la topologie TreeP. Dans le cas de 12, 18 et 24 noeuds, ceux-ci sont mappés en 6 noeuds serveurs et chacun entre eux manipule 2 (pour le cas de 12 noeuds) ou 3 (pour le cas de 18 noeuds) ou 4 (pour le cas de 24 noeuds) noeuds locaux.

En plus, ils ont effectué d'autres tests par diffusion des requêtes pour comparer avec le TreeP. Dans ce cas, la requête de recherche de n'importe quel noeud est envoyée au noeud serveur, si la connaissance n'est pas trouvée, le noeud serveur va diffuser cette requête à tous les autres noeuds dans le système (ce test simule la recherche sans le KM). Les résultats montrent qu'ils gagnent 3% à 10% en temps d'exécution qui est donné par le temps entre l'envoie de la requête et la réception des résultats de la recherche. Quoique

le chemin de communication de l'opération de diffusion est petit que celui dans le cas de la distribution à base de TreeP, la différence entre les deux exécutions est significative et les raisons sont :

- Le goulot d'étranglement au tour du noeud serveur dans l'opération de diffusion est important.
- La recherche de connaissances entières au lieu de méta-connaissances organisées par le KM prend plus du temps.
- Dans l'implémentation à base du TreeP, la recherche de méta-connaissances se fait à chaque noeud serveur le plus performant de son propre groupe.

En outre, les concepteurs de KM ont évalué la complexité des opérations de recherche et d'extraction de connaissances qui sont données respectivement par $O(\log M + Cs)$ et $O(\log m + Cl)$ avec M qui représente le nombre d'entrées de méta-connaissances dans le noyau de KM, m qui représente le nombre d'entrées de connaissances dans le KM local. Cs donne le coût du communication entre le noeud qui lance la requête de recherche de connaissance et le serveur contenant le répertoire de méta-connaissances et Cl donne le coût du communication entre le noeud qui lance la requête de recherche de connaissance et le noeud local contenant la connaissance.

2.5 Conclusion

Dans le présent chapitre, nous avons présenté l'architecture de Knowledge Map (KM) qui a comme objectif de gérer les connaissances extraites par les techniques de Data Mining dans les systèmes distribués. Cette approche facilite la visualisation des résultats d'extraction et supporte les tâches de Data Mining Distribué (DDM). D'après les estimations faites sur chaque composant et leur fonctionnalité nous pouvons dire que le système KM possède une architecture flexible et efficace en terme de représentation de connaissances. Il satisfait les besoins de gestion, de recherche des connaissances extraites dans les DDM. Cependant, ses concepteurs n'ont pas inclut le temps de construction des répertoires de méta-connaissances dans la complexité de KM alors que ce temps est très important et peut basculer les performances de cette architecture.

Chapitre 3

L'Architecture Knowledge Map à base d'Ontologies (KMO)

3.1 Introduction

Le système Knowledge Map (KM) que nous avons présenté dans le chapitre précédent est un système efficace et flexible de gestion, de recherche et d'extraction des connaissances en se basant sur les répertoires de méta-connaissances. Le seul inconvénient de ce système réside dans sa construction manuelle de ces répertoires en se basant sur l'intuition des utilisateurs ; ce qui pose le problème de subjectivité et de lenteur dans la construction des répertoires et les réponses fournies sont partielles car cette architecture n'utilise pas la sémantique des connaissances. D'où notre motivation de développer une nouvelle architecture de gestion de connaissances qui garde le principe de KM en automatisant la construction des répertoires par l'utilisation des ontologies car ces dernières présentent un outil de représentation et de structuration de connaissances très avancé.

Le présent chapitre contient deux sections, la première présente l'architecture Knowledge Map à base d'Ontologies (KMO) ainsi que l'algorithme de construction de ses répertoires de méta-connaissance et la deuxième section est consacrée pour l'implémentation de KMO selon le Tree P2P (TreeP) et ses algorithmes de mise-à-jour.

3.2 Knowledge Map à base d'Ontologies (KMO)

Pour répondre à l'objectif principal de notre architecture KMO qui consiste à organiser les connaissances d'une manière simple et efficace afin de faciliter leurs accès aux utilisateurs il faut les bien structurer.

L'architecture KMO s'inspire de KM [36] (voir FIG. 2.1) à laquelle nous ajoutons les deux composants : base d'ontologies et le moteur de recherche (FIG. 3.1) qui permettent d'améliorer la construction de ses répertoires de méta-connaissances qui sont des facteurs clés de cette architecture. Nous présentons dans ce qui suit les composants principaux de notre architecture KMO :

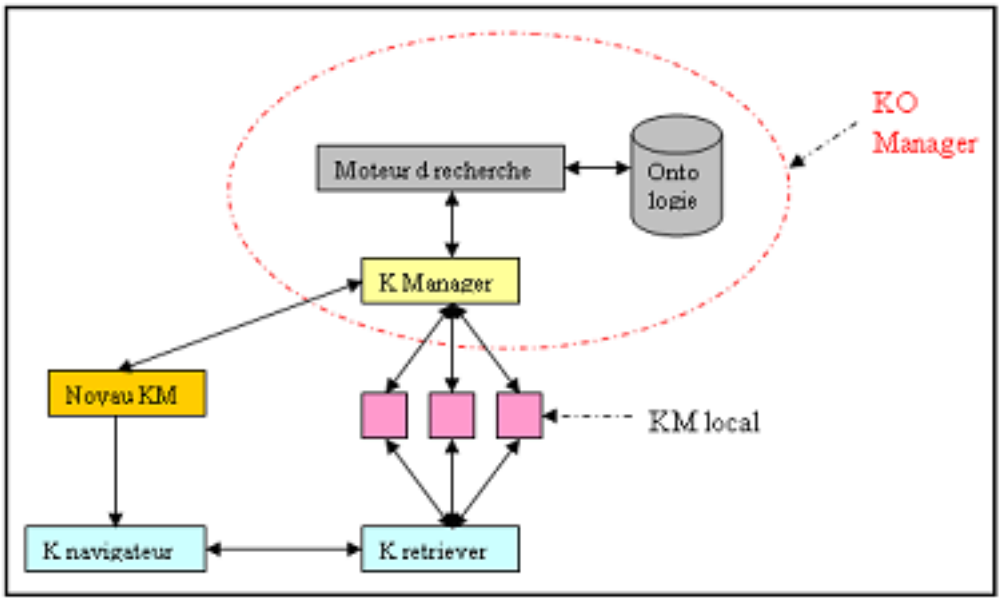

FIG. 3.1 – L'architecture de Knowledge Map à base d'Ontologies (KMO).

Knowledge Ontology Manager (KO Manager) : Ce composant est formé de trois parties principales appelées : Ontologie, Moteur de recherche et le Knowledge Manager (K Manager). L'ontologie est utilisée comme base pour la construction de répertoire d'arbres de concepts (voir FIG. 3.3) où l'entrée i est construite en utilisant l'ontologie *wine.owl* [39] et l'entrée j est construite en utilisant l'ontologie *food.owl* [39]. Chaque ontologie contient des concepts et des relations entre eux dans un domaine d'application en plus des instances obtenues par l'enrichissement des ontologies en utilisant des dictionnaires. Les concepts dans les ontologies sont structurés en une hiérarchie de plus général au plus spécifique en utilisant la propriété *is-a* (subsomption) et des relations entre les concepts comme montré dans l'exemple de la figure (FIG. 3.2). Dans cet exemple, les concepts sont représentés par des rectangles et les relations par des ellipses.

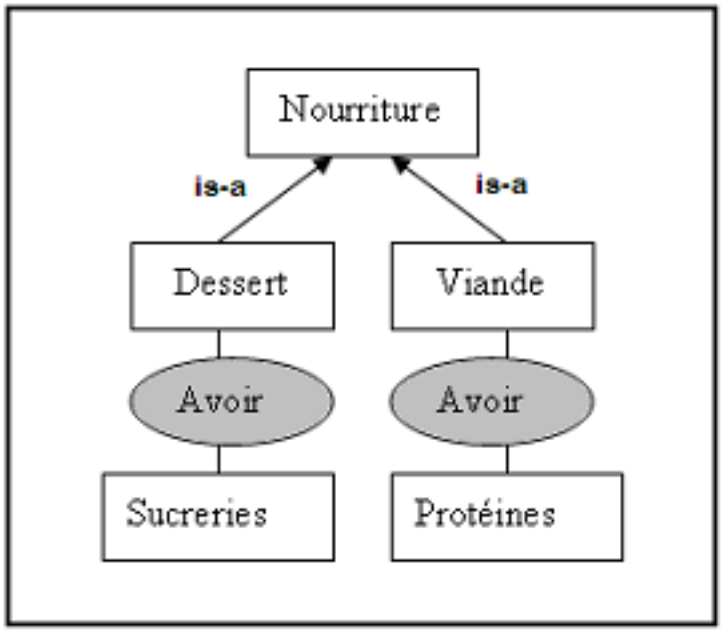

FIG. 3.2 – Exemple d'ontologie.

Le moteur de recherche représente l'algorithme de navigation dans l'ontologie pour extraire les concepts nécessaires suite à une requête de K Manager lors de la construction du répertoire d'arbre de concepts.

Le K Manager a comme fonction la création des différents répertoires de méta connaissances et ceux d'arbres de concepts par interaction avec le moteur de recherche. Il implémente l'algorithme *Construct ()* que nous détaillons ci-dessous. Ce composant permet aussi d'assurer la cohérence entre le noyau de KM et les KM locaux en faisant des mises-à-jour sur les répertoires de méta-connaissances et ceux d'arbres de concepts dans le cas où il existe un ajout ou une suppression de connaissances au niveau des KM locaux.

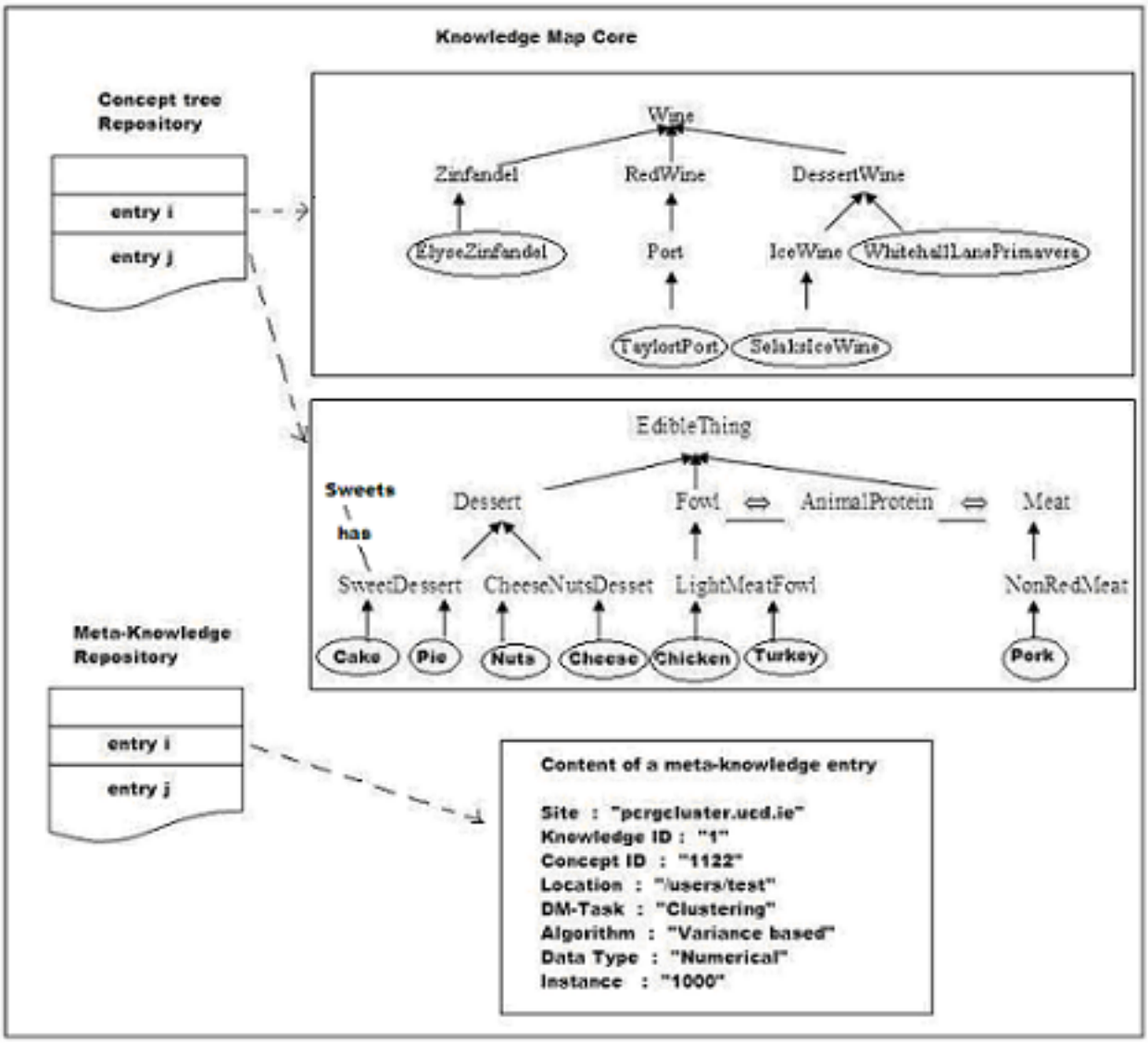

FIG. 3.3 – Noyau de KM.

Les hypothèses de l'algorithme *Construct()*

- Les ontologies sont sous forme hiérarchiques et enrichies par des individus ;
- Les bases de connaissances (KB) représentent les connaissances résultantes de l'application des techniques de DM comme le clustering, la classification, etc. Nous prenons par exemple, les règles d'associations obtenues par la classification des items qui sont souvent achetés ensembles par des clients qui sont sous forme : $item_1, item_2, ..., item_N \Rightarrow item_{i+1}, item_{i+2}, ..., item_M; (i \gg M)$ et $item_i$ qui correspond à une instance d'une ontologie.

Algorithm 1 Construct (construction du répertoire d'arbres de concepts)

1: *Inputs :* KB, ontologies de domaine ;
2: *Outputs :* Répertoire d'arbre de concepts ;
3: **Debut**
4: **Pour** chaque $règle_i$ dans KB **faire**
5: **Pour** chaque *item* $_j$ de la *règle* $_i$ **faire**
6: Déterminer le domaine D_k d'appartenance de l'*item* $_j$ (avec l'aide d'un expert de domaine) ;
7: Identifier l'ontologie O_k de domaine D_k (avec l'aide d'un expert de domaine) ;
8: Chercher l'$item_j$ dans l'ontologie O_k puis extraire le concept C_j qui le subsume (c'est-à-dire que $l'item_j$ est une instance de concept C_j) ainsi que toute la hiérarchie de concepts à partir de C_j jusqu'à la racine de l'ontologie O_k ;
9: Chercher dans l'ontologie O_k les concepts qui sont en relation ou équivalent à C_j et les extraire ;
10: **Fin pour**
11: **Fin pour**
12: **Fin**

Soient C le nombre de concepts dans l'ontologie de domaine utilisée, R le nombre de règles dans KB et I le nombre maximum d'items dans une règle, ainsi la complexité de l'algorithme *Construct ()* est $O(IR \log (C))$ et comme $I \ll R$ alors la complexité devient $O(R \log (C))$.

3.3 Implémentation de KMO

Dans cette section, nous allons discuter l'implémentation de notre architecture KMO qui est composée des éléments suivants : KM local, Noyau de KM, K Navigateur, K Retreiver et le KO Manager.

L'architecture KMO peut être implémenté en utilisant la topologie 1-n (1 serveur et n clients) dans laquelle les noeuds clients contiennent le KM local et le serveur gère le noyau de KM qui contient les répertoires de méta-connaissances des noeuds clients. Cependant, selon [1], le temps de l'opération de recherche est très grand par rapport à celui de l'extraction de connaissances à cause du nombre élevé des entrées des méta-connaissances qui peut générer un goulot d'étranglement important dans les systèmes distribués à grande échelle où un seul serveur est dédié à leurs gestions. Pour remédier à ce problème, nous avons utilisé la topologie TreeP [2] [3] qui offre les avantages suivants :

- Elle est très extensible, robuste, permet l'équilibrage de charge et facile à construire et à maintenir ;
- Elle exploite d'une manière efficace l'hétérogénéité des caractéristiques du réseau tout en limitant le surcoût introduit par l'entretien de recouvrement ;
- Elle est très résistante aux pannes qui peuvent survenir dans le réseau ;
- Elle réarrange le réseau P2P en topologie d'arbre ce qui lui offre des caractéristiques importantes comme la découverte de services cibles sans ajouter une autre couche d'abstraction au système middleware ;
- En outre, la structure des ontologies reflète exactement la structure de TreeP ce qui facilite leur distribution.

3.3.1 L'implémentation à base de TreeP

✍ La structure de TreeP

La topologie TreeP est organisée en arbre-hiérarchique qui possède un ensemble de noeuds physiques et un ensemble de noeuds virtuels. Les noeuds physiques (N_p) constituent les feuilles de la hiérarchie (niveau 0) et les noeuds virtuels (N_v) représentent une implémentation de noeuds physiques élus aux niveaux supérieurs selon plusieurs critères

comme la puissance de la CPU, la capacité de réseau, la largeur de la bande passante, la capacité de stockage, la charge de travail, etc. Ils assurent la fonction de contrôle des noeuds sous leur responsabilité. L'objectif principal de TreeP est de construire une topologie où les noeuds peuvent joindre ou quitter le système à tout moment, tout en gardant les caractéristiques globales de la topologie inchangées et en assurant les propriétés suivantes : cohérence, robustesse, performance et scalabilité.

Pour atteindre cet objectif, les noeuds sont organisés hiérarchiquement selon une topologie d'arbre B+ [40] dont le point fort réside dans les propriétés d'insertion et de suppression de noeuds. Ces propriétés ont une complexité de $(O \log_d (N))$ avec N qui représente le nombre de noeuds physiques) et laissent l'arbre équilibré (tous les chemins du noeud racine aux noeuds feuilles sont de même longueur). La hauteur h de l'arbre est logarithmique avec le nombre de noeuds N, elle est donnée par $h = \log_c(\frac{N+1}{2})$ [3] (avec $(2(\lceil \frac{2d}{3} \rceil + 1)^{h-1} - 1 \leq N \leq (2d+1)^h - 1)$ [41] et c qui représente la moyenne de nombre de fils par parent). Ainsi, le plus long chemin entre deux noeuds est en $(O \log_d (N))$ avec d qui représente le degré de l'arbre TreeP).

La figure (FIG. 3.4) [1] illustre un exemple d'une structure arborescente TreeP où l'ordre $d = 3$ et la hauteur $h = 4$. Les noeuds de même niveau sont groupés en sous réseaux appelés blocs, ainsi nous trouvons des blocs physiques au niveau 0 et des blocs logiques aux niveaux supérieurs. Chaque bloc est formé par au moins $\lceil \frac{2d}{3} \rceil$ noeuds et au plus $2d$ noeuds.

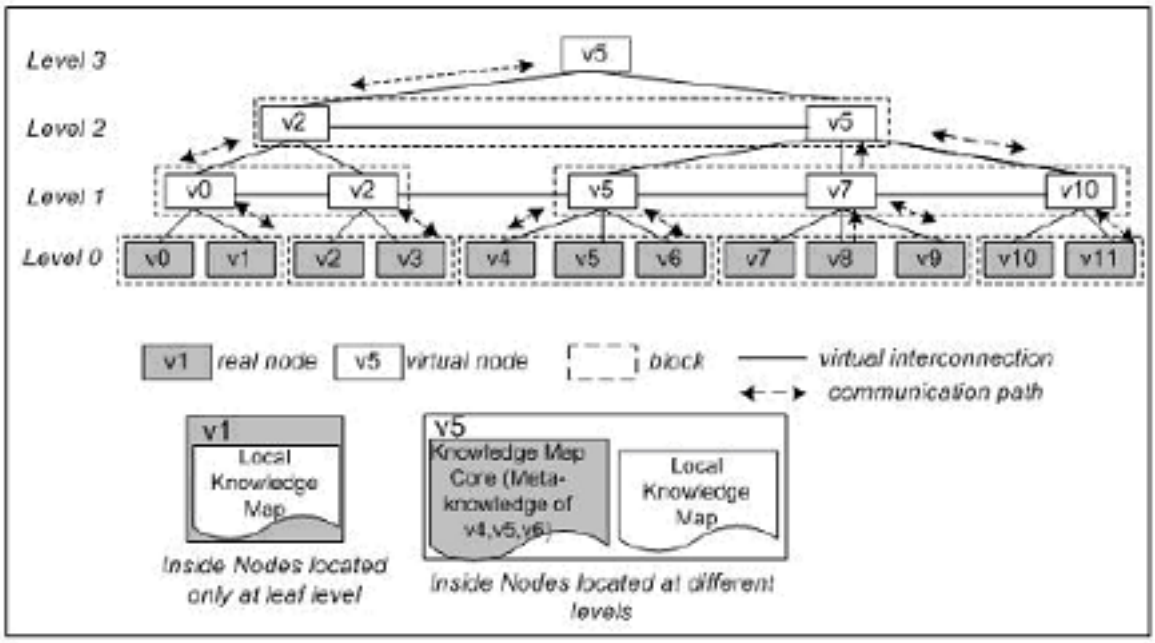

FIG. 3.4 – La topologie d'arbre représentant le TreeP.

✍ La construction de la hiérarchie TreeP

Pour construire la topologie TreeP, nous devons allouer de nouvelles tables de routage et créer de nouveaux chemins dédiés pour relier les noeuds. Le système fonctionne à l'aide de ces nouvelles tables de routages associées aux noeuds ; où un noeud ne peut envoyer/recevoir des messages que de/vers les noeuds qui lui sont adjacents [2].

Pour la topologie TreeP, chaque noeud (virtuel ou physique) a un identificateur unique ; ce qui donne un aperçu de sa position sur la topologie. A chaque noeud, il est assigné une paire de paramètres *(Id, w_i)* où *Id* est l'identifiant du noeud qui peut être une adresse *IP* ou un rang d'arrivée et w_i est le poids du noeud qui est calculé en fonction de sa puissance de calcul et son temps depuis son arrivée dans le système. *Id* est un identifiant unique pour localiser un noeud donné sur la hiérarchie. Il est donné par son nom d'origine auquel nous ajoutons son ancêtre à partir de la racine jusqu'aux feuilles de la topologie (voir FIG. 3.5). Donc l'*Id* peut être divisé en deux parties : le vrai nom du noeud et la partie qui lui est associée en fonction de sa position sur la hiérarchie. Il est mis-à-jour lorsque la structure de l'arbre change. Le poids w_i doit prendre en compte la puissance de traitement de chaque noeud et son temps de connexion depuis son arrivée dans le système. Les poids sont mis-à-jour dès que l'un des paramètres de noeud a changé. Les noeuds avec les plus grands poids sont les plus puissants. Les noeuds qui intègrent la hiérarchie à partir des feuilles de l'arbre sont promus aux niveaux supérieurs selon leurs caractéristiques.

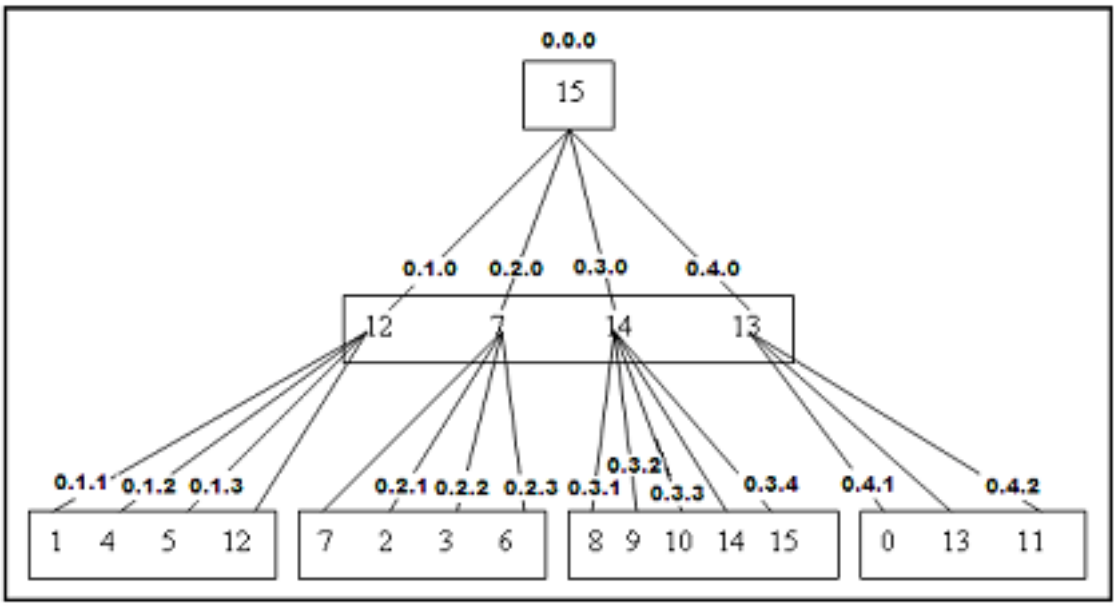

FIG. 3.5 – Identificateurs de noeuds dans le TreeP.

Au début, chaque noeud ne connaît que ses voisins. Un noeud donné que nous appelons la racine du système, déclenche la construction de cette hiérarchie. Il envoie un message contenant les caractéristiques de l'arbre à construire à l'ensemble de ses voisins. Chaque

noeud recevant ce message le transmet à son voisinage et contribue à la construction du système et ainsi de suite. Nous supposons que chaque noeud peut exécuter une des tâches élémentaires à la fois. L'algorithme [2] utilisé pour construire la hiérarchie est présenté ci-dessous.

Algorithm 2 Construction de la topologie TreeP

Début
  Compter le nombre de noeuds disponibles ($nn = N$) et les trier par l'ordre croissant selon les poids
  Tant que $nn > 1$ **faire**
    Compter le nombre des blocs $N_{bloc} = \lfloor \frac{\lfloor \frac{nn}{2} \rfloor}{2d} \rfloor + \lceil \frac{\lfloor \frac{nn}{2} \rfloor}{\lceil \frac{2d}{3} \rceil} \rceil$
    Si $nn < 2d$ **alors**
      Tout les noeuds appartiennent à $bloc_0$
      Sinon
        $\beta \leftarrow \lfloor \frac{nn}{N_{bloc}} \rfloor + 1$
        $\alpha \leftarrow N - N_{bloc} \lfloor \frac{nn}{N_{bloc}} \rfloor$
      Si $\alpha = 0$ **alors**
        $mon_{bloc} = \lfloor noeudrang/\beta \rfloor$
        Sinon
          Si $noeud_{rang} < (N_{bloc} - \alpha)\beta$ **alors**
          $mon_{bloc} = bloc \lfloor noeudrang/\beta \rfloor$
          Sinon
            $mon_{bloc} = bloc \lfloor noeudrang/\beta \rfloor - 1$
            Fin si
        Fin si
      Fin si
    Choisir un noeud virtuel pour le pavage
    Fin tant que
  Choisir la racine de l'arbre
Fin

Si les paramètres des noeuds ne changent pas pendant la phase de la création, le nombre de messages nécessaires pour la construction de la hiérarchie TreeP est $O\ (N \log\ (N))$[41].

Dans leurs simulations, les auteurs de [41] ont déduit que le coût de construction de l'arbre diminue avec l'augmentation de l'ordre de la topologie, quoique cet ordre a un impact sur la taille de la table de routage gérée par le noeud parent et par conséquent, il

faut trouver un compromis pour satisfaire les deux critères.

✍ L'insertion d'un noeud

L'insertion d'un noeud dans le système est une opération habituelle exécutée plusieurs fois. Un noeud qui arrive dans le système devrait être inséré dans un bloc, sans perturber l'équilibre de l'arbre et sans consommer les ressources du système. Un noeud entrant demande à la racine de l'arbre un service qui consiste à identifier le noeud auquel il doit appartenir. En fonction de son poids, le noeud va descendre dans l'arborescence en utilisant la stratégie *"largeur d'abord"* tout en commençant par le bloc de droite pour atteindre le bloc où il doit appartenir au niveau des feuilles. A chaque niveau au dessus des feuilles, un noeud virtuel est choisi par ses parents pour contenir la nouvelle entrée. A la fin de l'insertion du noeud, les autres noeuds appartenant à un même bloc mettent à jour leurs états ainsi que leurs tables de routage si nécessaire.

Soit v_k un noeud qui intègre l'arbre T, l'algorithme d'insertion [2] est donné ci-dessous :

Algorithm 3 Insertion de noeud v_k dans la topologie TreeP

```
1:  Début
2:      Niveau = h − 1
3:      Tant que (Niveau ≥ 0) faire
4:          Trouver la place droite pour v_k
5:          Niveau ← Niveau − 1
6:      Fin tant que
7:      Insérer v_k dans le bloc droit
8:      Tant que (le bloc est débordé) faire
9:          Créer un nouveau bloc
10:         Redistribuer les noeuds avec un de ses voisins
11:         Choisir un nouveau chef
12:         Choisir des chefs pour les blocs parents
13:         bloc ← bloc de père
14:     Fin tant que
15: Fin
```

D'après leur simulation, E.EDI et al 2006 [41] ont déduit que le coût d'insertion en fonction du nombre de messages envoyés diminue quand l'ordre de l'arbre est petit il est de l'ordre $(O(\log_d N))$.

✍ La suppression d'un noeud

La suppression d'un noeud dans la hiérarchie signifie que ce dernier quitte le système pour n'importe quelle raison. Ce noeud peut au plus appartenir à deux niveaux (sa représentation virtuelle et sa représentation physique). L'algorithme [2] suivant illustre les étapes de suppression de noeud v_k de l'arbre T.

Algorithm 4 Suppression de noeud v_k de la topologie TreeP

bloc=B_k , niveau=h-1
Tant que ($niveau > 0$) **faire**
  Si (B_k a un nombre de noeuds $< d$) **alors**
    Si (B_k a un voisin) **alors**
      Si (ce voisin a plus de $\lceil \frac{2d}{3} \rceil$ noeuds) **alors**
        Emprunter un élément de lui
        Sinon
          Diminuer d'un niveau de bloc parent
          Fusionner ces deux blocs
          Fin si
      Sinon
        Diminution d'un niveau tous les noeuds de bloc
        Diminution d'un niveau de bloc parent
        Fin si
    Sinon
      Si bloc==B'_k **alors**
        Élire un noeud de sous-arbre qui va être promue pour remplacer k
        Sinon
          bloc=bloc parent
          niveau=niveau-1
        Fin si
      Fin si
Fin tant que

Pour la suppression d'un noeud, le coût augmente avec l'ordre de l'arbre. Il possède une complexité de l'ordre ($O(\log_d(N))$) en nombre de messages envoyés pour équilibrer l'arbre. L'avantage de la topologie TreeP repose sur le fait que les noeuds des niveaux les plus élevés de la hiérarchie ont été promus en fonction de leurs performances de calcul et de leur stabilité ; donc les noeuds de niveaux supérieurs quittent rarement le système.

✍ **La recherche d'un noeud dans le TreeP**

L'algorithme de recherche *" look-up "* se base sur différentes tables de routages qui se trouvent sur les noeuds. Son but est de trouver un chemin entre deux noeuds distincts de TreeP. Du fait que les noeuds ont une vision globale du système, la recherche dans la topologie TreeP est basée sur la stratégie de recherche aveugle comme illustré par l'algorithme [2] suivant :

Algorithm 5 Recherche d'un noeud dans la topologie TreeP (look-up)

1: **Début**
2: **Si** le noeud recherché est sur votre table de routage **alors**
3: Arrêter de chercher
4: **Sinon**
5: Envoyer la demande à votre père
6: Envoyer la demande aux noeuds que vous connaissez à votre niveau
7: Envoyer la demande à votre enfants si et seulement si votre poids est plus grand que le poids du noeud recherché
8: **Fin si**
9: **Fin**

3.3.1.1 Mapping de KMO sur le TreeP

Cette nouvelle topologie TreeP est créée pour faciliter la recherche ainsi que l'acheminement des messages à partir des sources vers les destinations. Pour l'adapter à notre architecture KMO, nous associons les KM locaux aux noeuds physiques et les noyaux de KM ainsi que les KO Managers aux noeuds virtuels. Nous prenons un exemple pour illustrer TreeP ainsi que ses algorithmes de maintenance et celui de recherche de connaissances.

Soient les noeuds S_1, S_2, S_3, S_4, S_5, S_6 et S_7 des sites du réseau donné par la figure (FIG. 3.6) qui montre les répertoires de méta-connaissances de chacun de ces noeuds construits par l'algorithme *Construct ()* vu précédemment. Il est clair sur cette figure que les répertoires de méta-connaissances (exactement ceux d'arbres de concepts) de quelques noeuds couvrent ceux d'autres noeuds, ainsi pour construire le TreeP, nous faisons l'élection des noeuds couvrants comme des noeuds virtuels sur les noeuds couverts. Pour cet exemple, nous allons élire le noeud S_4 comme noeud virtuel sur les noeuds S_1 et S_3 car son répertoire d'arbres de concepts couvre ceux des noeuds S_1 et S_3, puis nous allons élire le noeud S_5 comme noeud virtuel sur le noeud S_2, ensuite le noeud S_6 est élu comme noeud virtuel sur le noeud S_7, nous passons au niveau 1 pour examiner les noeuds virtuels S_4,

S_5 et S_6. Nous constatons que le répertoire d'arbre de concepts du noeud S_6 couvre ceux des noeuds S_4 et S_5. Par conséquent, le noeud S_6 est élu comme noeud virtuel sur les noeuds S_4 et S_5. Le TreeP résultant est montré dans la figure (FIG. 3.7) dont la racine est le noeud S_6.

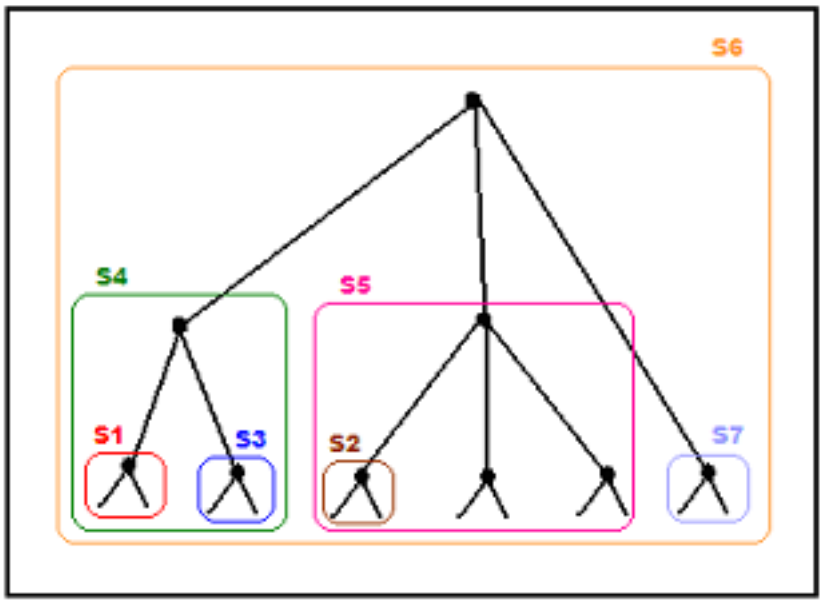

FIG. 3.6 – Répertoires de méta-connaissances des noeuds.

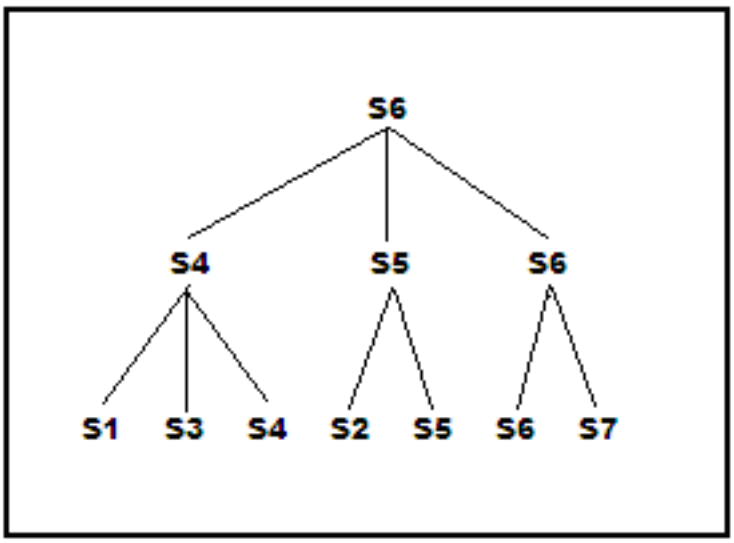

FIG. 3.7 – Exemple de TreeP.

✍ *Maintenance de TreeP et recherche de connaissances*

Les noeuds de TreeP sont en évolution permanente car ils peuvent acquérir de nouvelles connaissances ou supprimer des connaissances inutiles. En outre, il arrive qu'un noeud

quitte le système, (pour raison de panne, etc.) et la topologie doit être mise-à-jour. Les algorithmes de mise-à-jour sont :

Algorithm 6 Ajout de la connaissance $(x \Rightarrow y)$ dans le noeud S_k

1: **Début**
2: Ajout de la connaissance $x \Rightarrow y$ à la KB de S_k
3: **Si** x et y $\notin$ l'Ontologie **alors**
4: Extension de l'Ontologie par x et y
5: **Fin si**
6: **Si** x et y $\notin$ Porté (couverture) en $connaissance_{S_k}$ **alors**
7: S_k lance une demande de réorganisation de TreeP à son père et ses frères pour voire s'il y'a des noeuds qu'il peut couvrir en ayant la nouvelle connaissance $x \Rightarrow y$;
8: Mise-à-jour des Tables des noeuds concernés par l'organisation
9: **Fin si**
10: **Fin**

Soient D le nombre de noeuds dans le TreeP et d son degré, C le nombre de concepts dans l'ontologie. La complexité de l'algorithme d'ajout de connaissances est la somme des complexités $O(\log\ (C))$ et $O(\log\ (D))$ car ce dernier se compose de deux phases essentielles qui sont l'extension de l'ontologie (en $O(\log\ (C))$) et la réorganisation de TreeP ($O(\log\ (D))$). Comme $C \gg D$ alors la complexité est de l'ordre $O(\log\ (C))$.

Algorithm 7 Suppression d'une connaissance x dans un noeud S_k

1: **Début**
2: niveau=0 ; $site = S_k$; boolien=faux ; h=hauteur de TreeP
3: Supprimer x de la KB de S_k ;
4: **Tant que** niveau $\neq h$ et boolien = faux **faire**
5: site contactes père ($pere_{site}$) pour vérifier s'il a x ;
6: **Si** la réponse est non **alors**
7: site = $pere_{site}$;
8: $niveau = niveau + 1$;
9: **Sinon** boolien = vrai ;
10: **Fin si**
11: **Fin tant que**
12: **Si** boolien = faux **alors**
13: Suppression de la branche x de l'ontologie car là nous somme sûre qu'aucun noeud ne l'utilise ;
14: **Fin si**
15: **Si** S_k n'a plus de connaissances **alors**
16: Appeler l'algorithme de suppression de noeud S_k
17: **Sinon**
18: S_k contacte ses fils pour le couvrir ;
19: **Si** oui **alors**
20: Le fils de S_k devient son père ;
21: Réorganisation de sous-arbre de S_k ;
22: Mise-à-jour des Tables des noeuds ;
23: **Fin si**
24: **Fin si**
25: **Fin**

Cet algorithme se compose de deux étapes essentielles à savoir la vérification de l'utilisation de la connaissance par un autre noeud ($O(h)$) et la suppression ($O(\log_d(D))$) ou la réorganisation de TreeP (dans le pire cas reconstruction de TreeP $O(D\log(D))$), donc sa complexité est de l'ordre de $O(hD\log(D))$). Comme $D \gg h$ alors la complexité est donnée par $O(D\log(D))$.

Algorithm 8 Suppression de noeud S_k

Début
    Bk : le block physique de S_k ; Bk' son block logique ; $niveau_{Sk}$ le niveau le plus haut de S_k en commençant des feuilles, $pere_{S_k}$ le noeud père de noeud S_k ;
    niveau = 0 ;
    Si Bk = Bk' (ç-à-dire niveau $S_k = 0$) **alors**
        Suppression de S_k
        Si | Bk | = 1 **alors**
            $pere_{Sk}$ quitte le niveau 1 et rejoint le niveau 0 tout en gardant le même père ;
        Fin si
        Sinon (S_k est un noeud physique au même temps virtuel)
            Aller au niveau 0
            Tant que (niveau $\neq$ niveau S_k) **faire**
                Élire un noeud puissant pour le niveau+1
                $niveau = niveau + 1$
                Fin tant que
        Fin si
    Le noeud $pere_{Sk}$ envoie un message de mis-à-jour des Tables (père, porté en connaissances, fils, frères) à tous les noeuds concernés dans le TreeP ;
Fin

La complexité de cet algorithme est $O(\log_d (D))$ puisque la topologie en question est un TreeP [2].

Lors de la recherche de connaissances, les utilisateurs lancent l'application sans tenir en compte leur location. Les étapes de la recherche sont résumées dans l'algorithme suivant :

Algorithm 9 Recherche de connaissances dans le TreeP

1: **Début**
2: Sr : noeud qui lance la recherche de la connaissance ;
3: Sp : père de Sr ;
4: Sr envoie sa requête à Sp (par la primitive *Search()*)
5: Sp consulte son répertoire de méta-connaissance (MC) pour voir s'il existe un noeud dans son sous-arbre contenant la connaissance recherchée ;
6: **Si** la réponse est oui **alors**
7: Sp envoie les MCs de cette connaissance à Sr (par la primitive *Find()*)
8: **Si** Sr valide les MCs **alors**
9: Sp envoie à Sr les identifiants des noeuds contenants la connaissance par consultation de sa Table (père, connaissances, fils, frères) ;
10: Sr envoie sa requête à ces noeuds pour lui envoyer la connaissance (par la primitive *Retrieve()*) ;
11: **Fin si**
12: **Sinon** (la connaissance recherchée n'existe pas au niveau de Sp ou bien Sr n'a pas validé les MCs) **alors**
13: Sp envoie la requête à son père et ses frères et attend leur réponse pour répondre à Sr ;
14: **Fin si**
15: **Fin**

Pour cet algorithme de recherche de connaissances, la requête arrive dans le pire des cas jusqu'à la racine de TreeP car à ce niveau nous aurons sûrement une réponse sur les localisations de la connaissance recherchée et ceci a une complexité de $O(h)$ avec h la hauteur de TreeP et si tous les noeuds contiennent les connaissances recherchées alors le noeud *Sr* contacte tous les noeuds de TreeP en $O(D)$ pour extraire les connaissances, comme $D \gg h$ alors la complexité de l'algorithme de recherche de connaissances est de l'ordre $O(D)$.

3.4 Conclusion

Dans ce chapitre, nous avons présenté l'architecture Knowledge Map à base d'Ontologies. Nous avons développé un outil qui offre la possibilité d'organiser la connaissance extraite dans les systèmes distribués à grande échelle. Nous avons développé l'algorithme *Construct()* dont le but est d'intégrer les ontologies dans la construction des répertoires de méta-connaissances pour faciliter la gestion de la connaissance et représenter ces connaissances selon leurs rapports sémantiques et pour fournir un environnement viable pour les applications de DDM. Cet algorithme a comme entrées les ontologies de domaine ansi que les connaissances à organiser et a comme sortie un fichier écrit en language *OWL* qui représente le répertoire de méta-connaissances. En plus, nous avons développé des algorithmes de mise-à-jour de KMO pour garder sa cohérence et assurer son fonctionnement.

Notre architecture est validée par le calcul des complexités des algorithmes qui la manipulent qui sont logarithmique ou linéaire en nombre de messages qui circulent dans le réseau ainsi nous pouvons dire qu'elle est robuste.

Chapitre 4

Expérimentation et Evaluation de KMO

4.1 Introduction

Dans le présent chapitre, nous présentons le développement des composants principaux de notre architecture Knowledge Map à base d'Ontologies (KMO). Ainsi, ce chapitre est organisé en quatre sections. Nous présentons d'abord l'environnement de développement dans la première section puis nous définissons le scénario du test dans la deuxième section. La troisième section est consacrée à la présentation des résultats obtenus.

4.2 L'environnement de développement

Pour expérimenter l'architecture proposée, nous avons choisi de développer l'algorithme de construction de répertoires de méta-connaissances ainsi que celui de recherche de connaissances par utilisation du langage JAVA sous Windows. Ce choix est fait pour la simplicité de ce langage ainsi que le parallélisme d'exécution qu'il offre par utilisation des tâches threads qui simulent les noeuds du réseau dans notre architecture. Ce langage permet aussi la communication entre les threads par utilisation des sockets. Concernant la manipulation des ontologies ainsi que la construction des répertoires de méta-connaissances en OWL, nous avons utilisé l'API JENA que nous avons intégré sous JAVA par une simple configuration.

4.3 Présentation de scénario du test

Pour illustrer l'exécution de notre application, nous prenons un système distribué de dix noeuds $(S_0, ., S_9)$, chacun a une base de connaissance qui représente une classification des produits consommables souvent achetés ensembles. La figure 4.1 présente l'ontologie *food.owl* [39] adéquate pour la construction des répertoires de méta-connaissances.

La table (TAB. 4.1) résume les connaissances possédées par les noeuds du réseaux. Elles sont structurées sous forme de règles d'association qui ont une *prémisse* et une *conclusion*. Si nous prenons, par exemple, la première règle du noeud S_1 : "$Halibut, Flonder \Rightarrow Scrod$", elle signifie que si un client achète les items $Halibut$ et $Flonder$ alors forcément il achètera $Scrod$.

Le noeud	Les règles d'associations
S_0	SwordFish ⇒ Tuna ;
S_1	Halibut, Flonder ⇒ Scrod ; Scrod, Tuna ⇒ Swordfish ; Swordfish, Tuna ⇒ Flonder ;
S_2	Veal, Steak ⇒ RoastBeef ; GarlickyRoast ⇒ BeefCurry ; BeefCurry ⇒ Pork, GarlickyRoast
S_3	Clams, Mussels ⇒ Crab, Lobster ; Mussels, Crab ⇒ Lobster
S_4	Oysters, Crab, Peaches ⇒ Clams, SwordFish ; FetticineAlfredo ⇒ PastaWithWhiteClamsSauce ; Pork, Crab ⇒ Fradiovolo, GarlickyRoast ; Steak, RoastBeef, Tuna ⇒ BeefCurry, Cake ; Pizza, Pork ⇒ SpaghettiWithTomatoSauce, Fradiovolo ; Pork, Swordfish, Scrod ⇒ Flounder, Halibut ; Mussels, Lobster ⇒ Oysters, Veal ; Steak, RoastBeef ⇒ BeefCurry, Cake ; Cheese, Nuts ⇒ FraDiavolo, FettucineAlfRedo ; MixedFruit, Crab, Oysters ⇒ Bananas, Pie, Pizza ;
S_5	Pie ⇒ Cake ;
S_6	Cake, Pie ⇒ Cheese, Nuts ; Cheese, Cake ⇒ Nuts ;
S_7	SpaghettiWithTomatoSauce ⇒ Fradiovolo ; FetticineAlfredo ⇒ PastaWithWhiteClamsSauce ; SpaghettiWithTomatoSauce, FraDiavolo ⇒ FettucineAlfRedo, PastaWithWhiteClamSauce ;
S_8	Bananas, Peaches ⇒ MixedFruit ; MixedFruit, Peaches ⇒ Bananas ;
S_9	FetticineAlfredo, Cake ⇒ Pie ; Pizza, MixedFruit, PastaWithWhiteClamSauce ⇒ Cheese, Peaches, Bananas ; FraDiavolo, Nuts, Peaches ⇒ SpaghettiWithTomatoSauce ;

TAB. 4.1 – Connaissances de chaque noeud de réseau sous forme de règles d'associations.

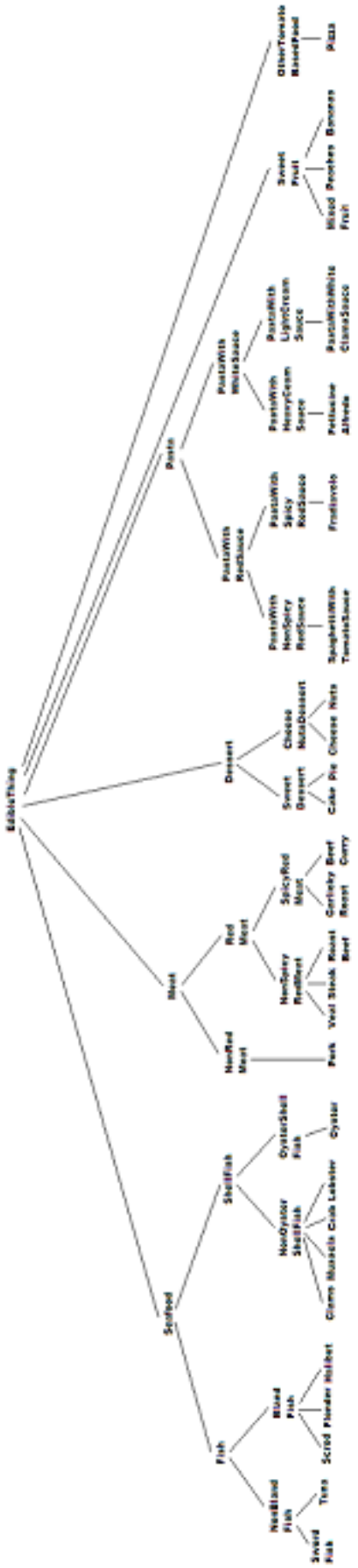

FIG. 4.1 – L'ontologie food.owl.

4.4 Les résultats du test

Pour commencer, nous structurons les noeuds de notre réseau sous forme d'un TreeP dont les noeuds virtuels implémentent le *noyau de KM* et les noeuds physiques implémentent les *KM locaux*. Pour se faire, nous utilisons un algorithme d'élection pour décider des noeuds qui vont jouer le rôle de noeuds virtuels.

Le répertoire de méta-connaissance d'un noeud (FIG. 4.3) reflète la partie de l'ontologie qui contient les connaissances de ce noeud. Il est construit à l'aide de l'algorithme *Construct ()* vu précédemment. Comme l'ontologie est structurée sous forme d'arbre, un noeud sera élu comme père d'un ensemble de noeuds si son répertoire de méta-connaissances inclue ou couvre les répertoires de ses fils. Nous commencons par la sélection des noeuds virtuels de niveau 1, le noeud S_0 a des connaissances sur *NonBlandFish* et le noeud S_1 a des connaissances sur *Fish* qui contient *NonBlandFish* et *BlandFich* (voir FIG. 4.3) alors le noeud S_1 sera élu comme noeud père (noeud virtuel) sur les noeuds S_0 et S_1. Pour le noeud S_5, il a des connaissances sur *SweetDessert* et le noeud S_6 a des connaissances sur *Dessert* qui contient *SweetDessert* et *CheeseNutsDessert* (voir FIG. 4.3) alors le noeud S_6 sera élu comme noeud père des noeuds S_5 et S_6. Le noeud S_2 a des connaissances sur *Meat*, le noeud S_3 a des connaissances sur *NonOysterShellFish* et le noeud S_4 a des connaissances sur *EdibleThing* qui contient *Meat* et *NonOysterShellFish* alors le noeud S_4 sera élu comme noeud père des noeuds S_2, S_3 et S_4. Une fois le niveau 1 de TreeP construit, nous passons à la construction de niveau 2, donc entre les noeuds S_1 et S_4 nous sélectionnons S_4 comme noeud père et entre les noeuds S_6 et S_9, S_9 est sélectionné comme noeud père. Ensuite, nous examinons le niveau 3 qui est le dernier niveau où nous sélectionnons le noeud S_4 comme père sur les noeuds S_4 et S_9 qui est la racine de TreeP illustré dans la figure FIG. 4.2.

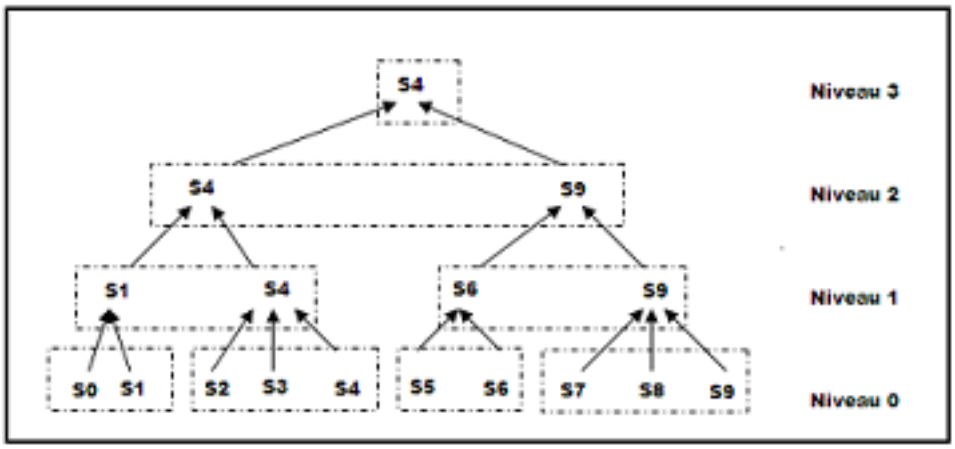

FIG. 4.2 – Le TreeP.

FIG. 4.3 – Répertoire de méta-connaissances de chaque noeud dans le TreeP.

Une fois le TreeP est construit, tous les noeuds impliqués doivent posséder une table qui reflète leur position dans ce dernier ainsi que les informations sur les noeuds dans leur sous-arbre comme les connaissances qu'ils contiennent, le père, les fils et les frères. Cette table est utilisée pour pouvoir situer les noeuds recherchés ainsi que la determination de leurs portés en connaissances (les connaissances qu'ils couvrent). Pour le noeud S_4, par exemple, la Table est de la forme suivante (TAB. 4.2) :

Noeud	Porté en connaissances	Père	Fils	Frères
S_0	NonBlandFish	S_1	/	S_1
S_1	BlandFish, NonBlandFish, Fish	S_4	S_0, S_1	S_4
S_2	Meat	S_4	/	S_3, S_4
S_3	NonOysterShellFish	S_4	/	S_2, S_4
S_4	EdibleThing	/	S_1, S_2, S_3, S_4, S_9	S_1, S_2, S_3, S_9
S_5	SweetDessert	S_6	/	S_6
S_6	Dessert, SweetDessert, CheeseNutsDessert	S_9	S_5, S_6	S_5, S_9
S_7	Pasta, PastaWithRedSauce, PastaWithWhiteSauce, PastaWithNonSpisyRedSauce, PastaWithSpisyRedSauce, PastaWithHeavyCreamSauce, PastaWithLightCreamSauce	S_9	/	S_8, S_9
S_8	SweetFruit	S_9	/	S_7, S_9
S_9	Dessert, Pasta, SweetFruit, OtherTomatoBasedFood	S_4	S_6, S_7, S_8, S_9	S_4, S_6, S_7, S_8

TAB. 4.2 – La Table de TreeP de noeud S_4.

Comme nous venons de le signaler, les répertoires de méta-connaissances des noeuds de TreeP sont écrits en langage OWL. Nous présentons un exemple de répertoire d'arbres de concepts du noeud S_1, il contient les connaissances couvertes par ce noeud à savoir : *Halibut*, *Flounder*, *Scrod*, *Tuna* et *Swordfish*. En plus de ces connaissances, ce répertoire contient les liens sémantiques entre ces dernières. Par exemple, il indique que *Scrod* est une instance du concept *BlandFish* qui est à son tour un concept subsumé par le concept *Fish*.

✍ *Le répertoire d'arbres de concepts du noeud* S_1

<rdf :RDF
xmlns :rdf="http ://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http ://krono.act.uji.es/Links/ontologies/file.owl#"
xmlns :owl="http ://www.w3.org/2002/07/owl#"
xmlns :xsd="http ://www.w3.org/2001/XMLSchema#"
xmlns :rdfs="http ://www.w3.org/2000/01/rdf-schema#" >
<rdf :Description rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#Halibut">
<rdf :type rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#BlandFish"/>
</rdf :Description>
<rdf :Description rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#Tuna">
<rdf :type
rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#NonBlandFish"/>
</rdf :Description>
<rdf :Description
rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#Fish">
<rdf :type rdf :resource="http ://www.w3.org/2002/07/owl#Class"/>
</rdf :Description>
<rdf :Description
rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#NonBlandFish">
<rdfs :subClassOf
rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#Fish"/>
<rdf :type rdf :resource="http ://www.w3.org/2002/07/owl#Class"/>
</rdf :Description>
<rdf :Description
rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#BlandFish">
<rdfs :subClassOf
rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#Fish"/>
<rdf :type rdf :resource="http ://www.w3.org/2002/07/owl#Class"/>
</rdf :Description>
<rdf :Description
rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#Flounder">
<rdf :type
rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#BlandFish"/>
</rdf :Description>
<rdf :Description

rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#Swordfish">
<rdf :type
rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#NonBlandFish"/>
</rdf :Description>
<rdf :Description
rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl#Scrod">
<rdf :type
rdf :resource="http ://krono.act.uji.es/Links/ontologies/file.owl#BlandFish"/>
</rdf :Description>
<rdf :Description rdf :about="http ://krono.act.uji.es/Links/ontologies/file.owl">
<rdf :type rdf :resource="http ://www.w3.org/2002/07/owl#Ontology"/>
</rdf :Description>
</rdf :RDF>

✍ *Recherche et extraction de connaissances dans KMO*

Lors de la recherche de connaissances dans le TreeP, le noeud concerné envoie sa requête (on suppose qu'il cherche *Crab*) sans soucis de la location de la connaissance recherchée si elle est locale ou distante. Comme le montre la figure (FIG. 4.4), nous cherchons d'abord les méta-connaissances nécessaires via la primitive *" Find "* (voir FIG. 4.4.a). L'utilisateur lance cette tâche via une interface graphique (FIG. 4.5). L'opération de recherche se compose de quatre étapes : (1) la requête est envoyée au noeud père dans le TreeP pour rechercher les méta-connaissances nécessaires. Puis celles-ci vont être extraites (2) et renvoyées au noeud requête (3) qui extrait les résultats comme objets de méta-connaissances (4) et l'utilisateur décide du niveau de méta-connaissance qui l'intéresse en sélectionnant la ligne appropriée puis clique sur le bouton *" Ok "* (voir FIG. 4.6). Nous extrayons les connaissances via la primitive *" Retreive "* (voir FIG. 4.4.b). Cette opération est aussi composée de quatre étapes : (1) les requêtes sont envoyées aux noeuds appropriés contenant la connaissance recherchée qui sont désignés par le noeud père en consultant la table TAB. 4.2 ; (2) extraire les connaissances trouvées dans chaque noeud ; (3) les renvoyer au noeud requête ; (4) extraire les connaissances comme objets de connaissances (voir FIG. 4.7).

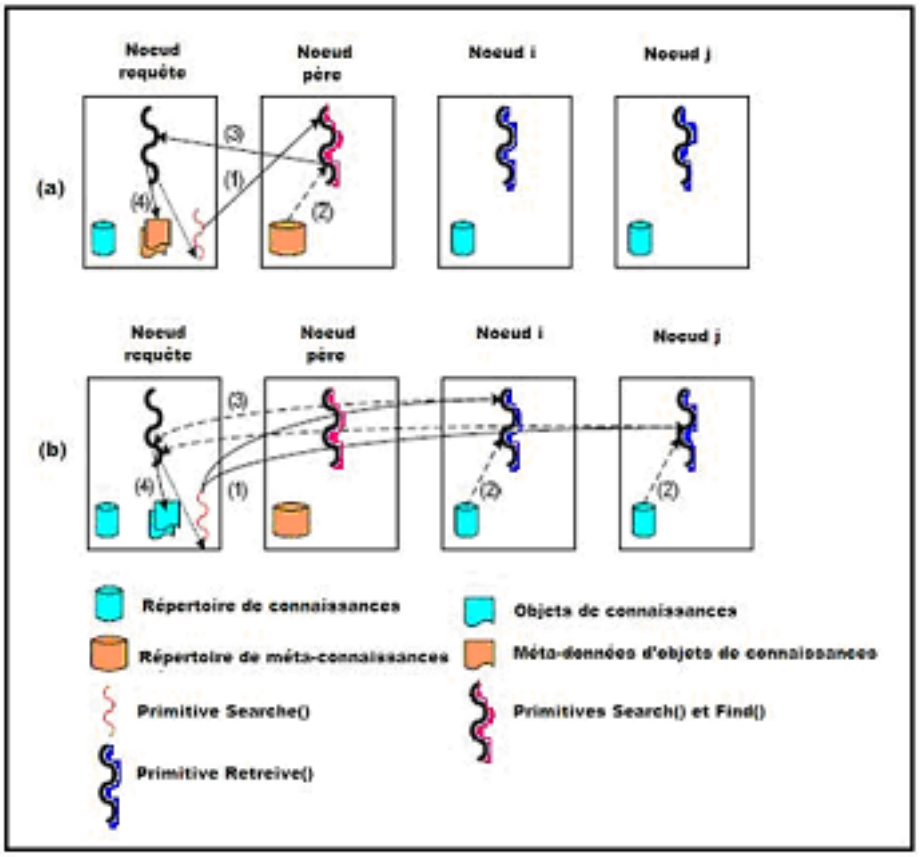

FIG. 4.4 – Exemple d'utilisation de KMO.

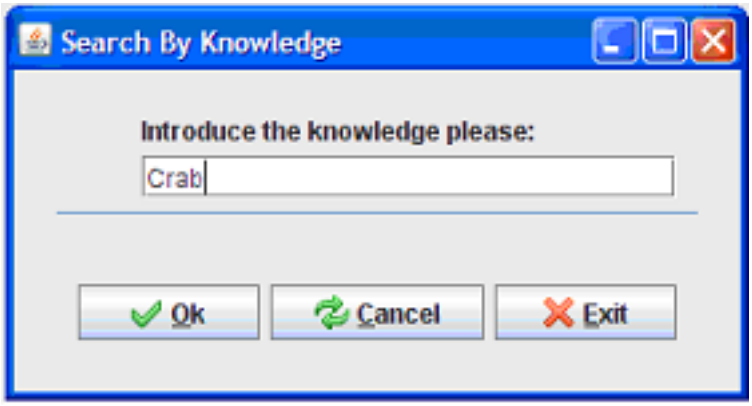

FIG. 4.5 – Capture d'écran de la recherche de connaissances.

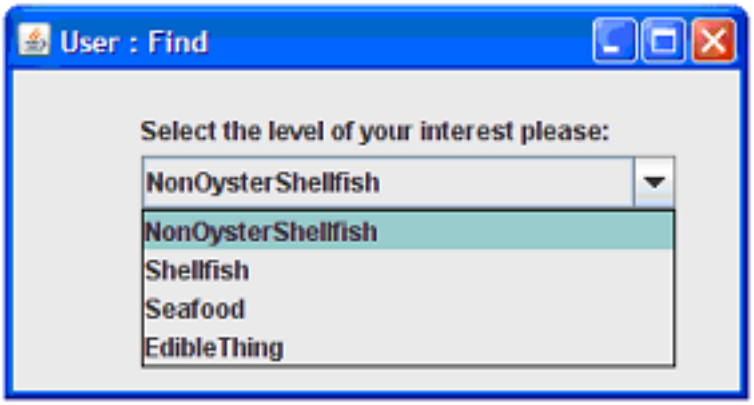

FIG. 4.6 – Capture d'écran de la recherche de méta-connaissances.

FIG. 4.7 – Capture d'écran de résultat de la recherche de connaissances.

4.5 Conclusion

Le composant principal, *KO Manager*, qui assure la qualité de la construction des répertoires de *noyau KM* n'existe pas dans l'architecture KM. Par conséquent, il est difficile de faire une comparaison entre les deux systèmes. Pour cela, nous validons cette nouvelle approche en évaluant les différents aspects de l'architecture du système en terme de gestion et de représentation des connaissances dans les collections de données distribuées complexes.

Nous avons vu dans la section 2 du chapitre précédent que les répertoires de méta-connaissances sont construits automatiquement à l'aide des ontologies de domaine. Cette opération présente l'avantage d'éviter toute subjectivité du problème posé dans le KM et le temps de construction des répertoires est réduit considérablement car ceci est réalisé de manière automatique. Ainsi, les mises-à-jour dans le KMO sont devenues très simples et rapides et la qualité des résultats est devenue plus satisfaisante puisque les répertoires offrent des liens sémantiques entre les connaissances. En plus de la sémantique, les répertoires de méta-connaissances utilisent la relation d'équivalence entre les connaissances. Ce qui n'est pas le cas de KM. Nous pouvons ainsi affirmer que notre approche fait une recherche exhaustive et retourne une réponse complète contrairement au système KM.

Notre outil KMO est destiné à la gestion des connaissances dans le framework ADMIRE [5] et ceci a nécessité sa distribution. Pour cela, nous avons opté pour la topologie TreeP car elle est très robuste [3]. Un autre apport de cette implémentation réside dans la vitesse de la recherche des connaissances et la réduction du nombre de messages qui circulent dans le réseau. Dans le cas le plus défavorable, un noeud communique avec *h-1* noeuds (avec h la hauteur de TreeP qui est logarithmique en nombre de noeuds de TreeP) pour connaître la position des connaissances recherchées.

Conclusion et Perspectives

Le Data Mining a attiré beaucoup d'attentions dans l'industrie de l'information ces dernières années, en raison de la grande disponibilité de quantités énormes de données et la nécessité de les convertir en informations utiles et en connaissances. Les informations et les connaissances acquises peuvent être utilisées dans divers domaines.

Des outils d'extraction des données sont développés pour permettre l'analyse de données et la découverte de modèles de données. La grande quantité et l'hétérogénéité des modèles résultants nécessitent un système de gestion qui offre aux utilisateurs un accès rapide, assure des résultats de bonne qualité ainsi qu'une utilisation efficace de la connaissance découverte. Ce système a même un impact positif sur la phase de pré-traitement car il permet aux utilisateurs d'avoir une réflexion sur les données à sélectionner, des techniques de DM à choisir ainsi que de la précision de la connaissance produite.

Dans la littérature, il existe plusieurs approches pour la gestion de connaissances parmi lesquels nous pouvons citer le framework d'extraction intensionnelle et la carte de connaissance pour la communauté virtuelle qui ont comme but d'aborder les problèmes de gestion de la connaissance dans le domaine de DM centralisé. Les projets qui traitent les problèmes de DDM à grande échelle sont le K-Grid et le KM. Le premier projet fournit une manière de gérer la connaissance mais pas l'intégration et la coordination des résultats. Par contre, le second projet règle ce problème et présente un système efficace qui satisfait les besoins de gestion, de recherche et d'extraction des connaissances en se basant sur les répertoires de méta-connaissances. Son inconvénient réside dans sa construction manuelle de ces répertoires en se basant sur l'intuition des utilisateurs ; ce qui pose le problème de lenteur d'une part et la qualité des répertoires qui est réduite car ces derniers ne possèdent pas vraiment une sémantique. Pour remédier à ce problème, nous avons proposé une nouvelle architecture de gestion de connaissances (KMO) qui garde le principe de KM en automatisant la construction des répertoires par l'utilisation des ontologies. Pour sa distribution, nous avons opté pour la topologie TreeP puisqu'elle est très robuste et facile

à construire et à maintenir en plus de la rapidité qu'elle offre dans l'accès aux différents noeuds.

Nous avons développé un algorithme de construction de répertoires de méta-connaissances qui a comme entrées des ontologies de domaine (enrichies par des individus en utilisant des dictionnaires) et les connaissances à organiser et comme sortie un fichier OWL contenant les liens sémantiques (méta-connaissances) de nos connaissances. Les répertoires de méta-connaissances constituent le noyau de l'architecture à base ontologique proposée. Nous avons également développé des algorithmes de recherche de connaissances ainsi que ceux de mise-à-jour de l'architecture et nous avons calculé la complexité de tous les algorithmes proposés qui sont entre logarithmes et linéaire. Nous montrons ainsi la robustesse de notre approche qui offre une amélioration dans le temps de création des répertoires de méta-connaissances. De plus, nous introduisons dans notre approche une représentation sémantique des connaissances

Nous avons implémenté l'algorithme de construction des répertoires de méta-connaissances ainsi que celui de la recherche de connaissances sous JAVA et l'API JENA et nous avons présenté les résultats d'implémentation des algorithmes ainsi qu'une validation par un scénario sur l'extraction de connaissances dans le domaine commercial représentant les produits souvent achetés ensemble.

Il est intéressant de poursuivre le développement d'une telle architecture pour implémenter les algorithmes de mise-à-jour.

Notre architecture Knowledge Map à base d'Ontologies (KMO) présente des limitations dues au problème de ramassement de données réelles (les connaissances résultantes d'application de techniques de Data Mining) pour valider le système significativement ce qui nous a poussé à la construction d'un ensemble artificiel qui est un peu réduit. En plus de ceci, nous marquons le manque d'ontologies qui sont compatibles avec la version 2.3 de l'API JENA qui est disponible pour le téléchargement, et nous ne pouvons pas nous permettre de développer nos propre ontologies à cause de la contrainte du temps.

Une autre lacune de l'implémentation de notre architecture réside dans sa lourdeur qui est due à son implémentation sur un seul PC où nous avons simulé les noeuds de TreeP par des threads sous JAVA qui utilisent des sockets pour se communiquer. Nous pouvons résoudre ce problème facilement si nous avons une grille de PC en instalant chaque thread (qui implémente les composants de KMO) sur un noeud de la grille.

Bibliographie

[1] N-A. Le-Khac, Lamine M. Aouad and M-T. Kechadi, "Handling large volumes of mined knowledge with a self-reconfigurable topology on distributed systems", IEEE Seventh International Conference on Machine Learning and Applications (IEEE ICMLA 2008) , December 11-13, 2008, CA, USA.

[2] E. EDI, M-T. Kechadi, and R. McNulty. "TreeP : A Self-Reconfigurable Topology for Unstructured P2P Systems". LNCS on State-of-the-Art in Scientific & Parallel Computing, Vol. 4699 p.1136-1146, 2007.

[3] B. Hudzia, M-T. Kechadi, and A. Ottewill. "TreeP : A Tree-Based P2P Network Architecture". IEEE, International Workshopon Algorithms, Models and tools for parallel computingon heterogeneous networks (HeteroPar05), Boston, Massachusetts, USA, September 27-30, 2005.

[4] N-A. Le-Khac, Lamine M. Aouad and M-T Kechadi, "Distributed Knowledge Map for Mining Data on Grid Platforms". IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.10, October 2007.

[5] N.A. Le-Khac, Lamine M. Aouad and M-T. Kechadi, "Toward a Distributed Knowledge Discovery on Grid Systems", Chapter in "Emergent Web Intelligence" Book, published by Springer Verlag in the series Studies in Computational Intelligence, 2009, ISBN xxxx-xxxx.

[6] Jiawei Han and Micheline Kamber, "Data Mining Concepts and Techniques", Book, Second Edition, 2006.

[7] P. Charbonnaud , "Représentation des connaissances", Disponible sur : http ://www.enit.fr/charbonnaud/Tsymbolique/4IA.pdf, Dernier accès le 08 Janvier 2011.

[8] J.-M. KARKAN & G. TJOEN, SYSTEMES EXPERTS Un nouvel outil pour l'aide à la décision, MASSON, Bruxelles, 1993.

[9] J. L. Crowley, "Représentation Structurée de la Connaissance",Disponible sur : http ://www-prima.imag.fr/jlc/Courses/2007/ENSI2.SIRR/ENSI2.SIRR.S6.pdf, Dernier accès le 08 Janvier 2011.

[10] Représentation des connaissances, Disponible sur : http ://postit.blog-cao.com/files/2010/11/04-Repr%C3%A9sentationdesconnaissances.pdf , Dernier accès le 08 Janvier 2011.

[11] W. McCulloch & W. Pitts, A logical calculus of the ideas immanent in nervous activity. 1943.

[12] S.K. Gupta, V. Bhatnagar and S.K. Wasan. "A propasal for Data Mining Management System". Integrating Data Mining and Knowledge Management Workshop, IEEE ICDM, 2001.

[13] M. Cannataro, D. Talia and P. Trunfio, "Distributed Data Mining on Grids : Services, Tools, and Applications", IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B : CYBERNETICS, VOL. 34, NO. 6, DECEMBER 2004.

[14] I.Foster, C.Kesselman, "Globus : A Metacomputing Infrastructure Toolkit", Disponible sur : http ://cseweb.ucsd.edu/ carter/260/globus.pdf, Dernier accès le 04 Juillet 2010.

[15] F. Lin and C.M. Hsueh. "Knowledge map creation and maintenance for virtual communities of practice". Journal of Information Processing and Management, vol. 42, 2006, 551-568.

[16] L.-f. Chien, "PAT-Tree-Based Keyword Extraction for Chinese Information Retrieval", Proceedings of the 1997 ACM SIGIR, Philadelphia, PA, USA, pp. 50-58.

[17] D. E. Knuth, "The Art of Computer Programming : Sorting and Searching", Vol. 3. Addison-Wesley, Mass., 1973.

[18] R. GRUBER, "A translation approach to portable ontology specifications", Knowledge Acquisition, 5(2) :199-220, 1993.

[19] T.Broekstra, M.Klein, S.Decker and al, "Enabling Knowledge representation on the Web by rdf Schema", In Proceeding of the 10th International World Wide Web Conference, Hong Cong, China, May 2001.

[20] M. HEMAM, "Un processus de développement d'ontologies dans le cadre du Web Sémantique", Mémoire, Algérie, 2005.

[21] USCHOLD M. & KING M., "Towards a methodology for building ontologies", in Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI'95, 1995.

[22] Charlet J., Bachimont B. et Troncy R. "Ontologies pour le Web Sémantique", In Le Web sémantique, CHARLET J., LAUBLET P. et REYNAUD C. (Ed.), Hors série de la Revue Information - Interaction - Intelligence (I3), 4(1), Cépaduès, Toulouse, 2004.

[23] KASSEL G., "OntoSpec : une méthode de spécification semi-informelle d'ontologies", in Actes des journées francophones d'Ingénierie des Connaissances (IC'2002), pages 75-87, 2002.

[24] F.FÜRST, "L'ingénierie ontologique", Institut de Recherche en Informatique de Nantes, RAPPORT DE RECHERCHE No 02-07 Octobre 2002.

[25] P.Lando, "Conception et développement d'applications informatiques utilisant des ontologies : application aux EIAH", 1res Rencontres jeunes chercheurs en EIAH, RJC-EIAH'2006.

[26] T.HOUACINE, "Construction et exploitation d'une ontologie dans le domaine de lutte antiacridienne", Algérie, 2008.

[27] L'ontologie DOLCE, Disponible sur : http ://www.loa-cnr.it/DOLCE.html, Dernier accès le 17 Juin 2010.

[28] L'ontologie SUMO, Disponible sur : http ://suo.ieee.org, Dernier accès le 17 Juin 2010.

[29] R.Mizoguchi et J.Bourdeau, "Using Ontological Engineering to Overcome Common AI-ED Problems.", International Journal of Artificial Intelligence in Education, IJAIED. vol. 11, no 2, 2000.

[30] P.ZWEIGENBAUM, "Encoder l'information médicale : des terminologies aux systèmes de représentation des connaissances", in Innovation stratégique en information de santé (ISIS) (2-3), pages 27-47, 1999.

[31] T.GRUBER & G.OLSEN, "An ontology for engineering mathematics", in J. DOYLE F. S.&TORANO P., eds., Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, Morgan-Kauffmann, 1994.

[32] M.GRUNINGER & M.S.FOX, "Methodology for the design and evaluation of ontologies", in Proceedings of the Workshop on Basic Ontological Issues on Knowledge Sharing, IJCAI'95, 1995.

[33] J.Cardoso, A.P.Sheth, "PROGRAMMING THE SEMANTIC WEB". In : "Semantic Web Services, Processes and Applications", Springer, 2006, Volume 3, Part III, 351-380, DOI : 10.1007/978-0-387-34685-4_14.

[34] T.Berners-Lee, J.Hendler, et al. "The Semantic Web", Scientific American. May 2001.

[35] S.Decker, S. Melnik, et al, "The Semantic Web : The Roles of XML and RDF", Internet Computing 4(5) : 63-74, 2000.

[36] N-A. Le-Khac, Lamine M. Aouad and M-T. Kechadi, "Knowledge Map : Toward a new approach supporting the knowledge management in Distributed Data Mining", KUI, IEEE The Third International Conference on Autonomic and Autonomous Systems ICAS 2007, Computer Society Press, Athens, Greece, June, 2007

[37] N.A. Le-Khac, Lamine M. Aouad and M-T. Kechadi,"An Efficient Knowledge Management Tool for Distributed Data Mining Environments",International Journal of Computational Intelligence Research. ISSN 0974-1259 Vol.5, No.1 (2009), pp. 5–15.

[38] L.M. Aouad, N-A. Le-Khac and M-T. Kechadi."Variance-based Clustering Technique for Distributed Data Mining Applications", International Conference on Data Mining (DMIN'07), USA, 2007 (conference style).

[39] R.Berlanga, "Temporal Knowledge Bases Group Universitat Jaume I", Disponible sur : http ://krono.act.uji.es/Links/ontologies, Dernier accès le 29 Mars 2010.

[40] D. Comer, "Ubiquitous B-tree". ACM Computing Survey, Vol.11, No.2, 1979, 121-137 (journal style).

[41] E.EDI, M-T.Kechadi, and R.Mcnulty, "TreeP : A Self-reconfigurable Topology for Unstructured P2P Systems", Workshop on State-of-the-Art in Scientific & Parallel Computing (PARA'06), Ume{å}, Sweden, June 18-21, 2006.