# Apport d'une ontologie du domaine pour affiner une requête à l'aide d'un treillis de Galois


B. SAFAR, H. KEFI

*Université Paris-Sud,*
*CNRS (LRI) & INRIA (Futurs)*
*LRI, Bâtiment 490*
*91405 Orsay cedex, France*
Email : {safar,kefi}@lri.fr
Tél : + 33 1 69 15 64 94 Fax : + 33 1 69 15 65 86



**Résumé**

Dans cet article, nous étudions comment aider un utilisateur qui effectue une recherche dans un entrepôt thématique à affiner sa requête quand celle-ci retourne trop de réponses. En utilisant une ontologie du domaine et un ensemble de ressources annotées avec les termes de cette ontologie, nous montrons comment utiliser un treillis de Galois pour élaborer, en interaction avec l'utilisateur, une requête plus précise qui réponde mieux à ses attentes.



**Abstract**

In this paper we study how to help user to refine his query when the search for documents in ressources has produced too many answers. Using a domain ontology and a set of ressources indexed with terms of this ontology, we show how we can use the immediate readability and intelligibility of the Galois Lattice clustering structure without paying the cost of its whole building. We also show how to build a refined query in interaction with the user.

# 1 Introduction

Quand il effectue une recherche d'information sur le web ou dans une base de données, l'utilisateur d'un moteur de recherche est bien souvent submergé par la masse de réponses retournées. Pour répondre à ce problème, de nombreux travaux ont portés sur le classement des réponses retournées en fonction de leur pertinence supposée ou sur leur regroupement en classes de réponses proches (clusters).

Dans cet article, nous étudions comment affiner la requête de l'utilisateur quand celle-ci obtient trop de réponses. Nous voulons l'aider à trouver les termes pertinents à rajouter à sa requête pour que celle-ci obtienne un nombre de réponses plus acceptable. Nous présentons pour cela OntoRefiner, un système qui crée des classes à partir des réponses obtenues et permet ensuite à l'utilisateur d'affiner ces classes de façon interactive. Ce système l'aide ainsi à se focaliser sur le sous-ensemble de réponses le plus pertinent et lui permet d'affiner sa requête initiale.

OntoRefiner est utilisé dans un entrepôt de données thématique. Le thème de l'entrepôt est décrit dans une ontologie du domaine composée d'une hiérarchie de termes ou mot-clés. L'entrepôt contient un ensemble de ressources (documents, services ou bases de données) relatives au domaine et chaque ressource est décrite par un ensemble de termes issus de l'ontologie. De même, l'utilisateur exprime ses requêtes au moteur de recherche associé à l'entrepôt en utilisant les termes de l'ontologie. Les ressources retournées par le moteur sont celles dont la description vérifie les termes de la requête posée.

Les entrées de l'algorithme de construction de classes sont l'ontologie du domaine, la requête posée et les descriptions des réponses retournées par le moteur. A partir de ces entrées, OntoRefiner construit des sous-ensembles grossiers et identifie dans chaque sous-ensemble des directions possibles de spécialisation. L'utilisateur peut alors choisir le sous-ensemble le plus intéressant ainsi qu'une direction de spécialisation. Le processus de construction de classes reprend sur le sous-ensemble choisi et se poursuit jusqu'à ce que l'utilisateur soit satisfait du résultat.

Le plan de l'article est le suivant. Nous présentons tout d'abord en section 2 l'ontologie du domaine et la façon dont nous l'utilisons pour faire apparaître de multiples points de vue dans une description de ressource. La méthode de classification par construction de treillis de Galois sur laquelle notre algorithme est basé et son intérêt pour l'affinement de requête sont présentés en section 3. Nous montrons ensuite en section 4 comment notre algorithme utilise l'ontologie du domaine pour ne construire que des fractions du treillis de Galois sans perdre son intérêt.

# 2 Ontologie et description des ressources

L'ontologie du domaine est décrite par une hiérarchie de noms de classe (noté $\mathcal{D}_h$) et représentée par un ensemble de règles de la forme $C_1 \rightarrow C_2$, où $C_1$ et $C_2$ sont des noms de classe ou termes. Par exemple dans le domaine des produits de tourisme, nous utilisons des termes comme AccomodationPlace, Localization, Equipment, Activities et la figure 1 montre un fragment de la hiérarchie décrivant le terme Localization selon différents points de vue, comme les aspects physiques ou géographiques.

Les deux définitions suivantes établissent la notion de généralisant d'un terme.

**Definition 2.1:** *Un* **généralisant direct** *d'un terme c est un terme c' tel que la règle c $\rightarrow$ c' appartient à $\mathcal{D}_h$. c est une spécialisation directe de c'.*

Un terme peut avoir plusieurs généralisants et plusieurs spécialisations.

**Definition 2.2:** *c' est un* **généralisant** *d'un terme c si c' est généralisant direct de c ou s'il existe c" tel que c' est un généralisant direct de c" et c" est un généralisant de c.*

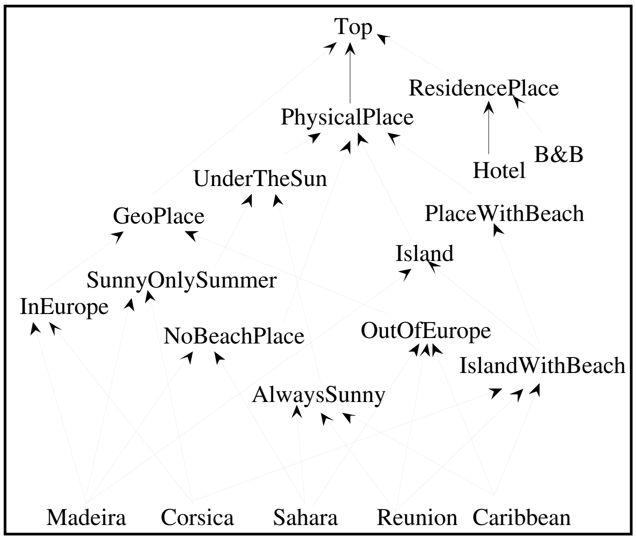


FIG. 1 – *Une fraction de la hiérarchie de termes*

**Exemple 1** *Les généralisants directs de Reunion sont AlwaysSunny, OutOffEurope et IslandWithBeach alors que ses autres généralisants sont PlaceWithBeach, Island, GeoPlace, etc. Les spécialisations directes de UnderTheSun sont SunnyOnlySummer et AlwaysSunny et ses autres spécialisations sont Corsica, Sahara, etc.*

Les ressources sont supposées être indexées[1] par les termes les plus précis de l'ontologie du domaine qui les caractérisent. En utilisant la hiérarchie de la figure 1, une ressource portant sur un hôtel situé à la Réunion aura pour description l'ensemble de termes suivant {Hotel, Located, Reunion}.

Avant d'effectuer la classification des ressources retournées, la description de chacune de ces ressources est enrichie, en utilisant la hiérarchie, de tous les généralisants des termes qui la composent. Ce mécanisme, appelé **saturation**, est défini comme suit :

**Definition 2.3:** *Sat($C$) est le* **saturé** *de l'ensemble de termes $C$, si pour chaque terme c de $C$, tous les généralisants de c sont dans Sat($C$).*

**Exemple 2** *Avec la hiérarchie de la fig.1, pour C = {Hotel, Located, Reunion}, Sat(C) = {Hotel, ResidencePlace, Located, Reunion, AlwaysSunny, UnderTheSun, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island, GeoPlace, PhysicalPlace}.*

Réciproquement, nous définissons la **désaturation** d'un ensemble $C$ comme le processus qui retire de cet ensemble tous les généralisants dont les spécialisations sont dans $C$.

**Definition 2.4:** *DeSat($C$) est le* **désaturé** *de l'ensemble de termes $C$, si pour chaque terme c de DeSat($C$), c est dans $C$ et il n'existe pas de c" dans $C$ tel que c"soit une spécialisation de c.*

L'objectif de cette étape de saturation est de faire apparaître des éléments communs entre des descriptions qui pourraient sinon ne pas en avoir. Ainsi dans une comparaison terme à terme, l'intersection des deux descriptions suivantes {Madere} et {Réunion} est vide alors que travailler sur les saturés fait émerger les généralisants communs {UnderTheSun, Island}. La saturation permet aussi de faire apparaître des similarités à un niveau d'abstraction plus élevé qu'entre les descriptions initiales (en utilisant les termes les plus généraux des descriptions) et/ou selon les différents point de vue pertinents pour le thème que recouvre implicitement un terme et qui sont explicités dans l'ontologie. Remarquons en effet que l'ontologie est orientée : nous nous intéressons au domaine du tourisme, et

1. Nous ne traitons pas ici le problème de l'indexation du contenu de chaque ressource. Nous supposons que cette tâche a déjà été réalisée par ailleurs, automatiquement ou manuellement.

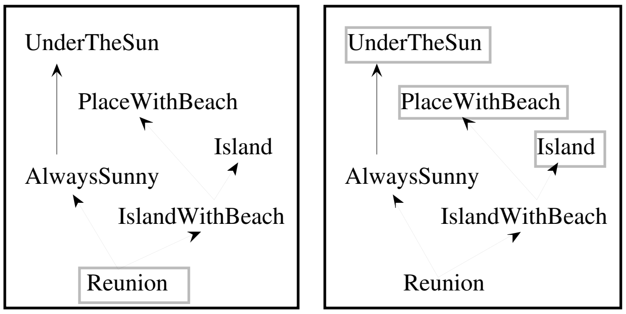


FIG. 2 – *fig.2a : Desat(C), fig.2b : mgt(C)*

les caractéristiques explicitées pour chaque terme sont celles qui sont pertinentes pour ce domaine, en terme de loisir (ici, pour les lieux, ce qui fait qu'on peut pratiquer ou pas des activités nautiques, se détendre au soleil, ...). Dans un autre contexte on pourrait s'intéresser aux points de vue historique, géologique, politique.

Dans la recherche des termes partagés par plusieurs descriptions, nous travaillerons donc sur les descriptions non désaturées et nous nous focaliserons sur les termes les plus généraux car ces termes ont plus de chances d'être communs à plusieurs descriptions.

**Definition 2.5:** *mgt($C$) est l'ensemble des* **termes les plus généraux** *de l'ensemble de termes $C$ si pour chaque terme $t$ de mgt($C$), $t$ est dans $C$ et il n'existe pas de $t'$ dans $C$ tel que $t'$ soit un généralisant de $t$.*

**Exemple 3** *Dans la figure 2, en utilisant la hiérarchie fig.1, pour $C$ = {Reunion, AlwaysSunny, UnderTheSun, IslandWithBeach, PlaceWithBeach, Island}, DeSat($C$) = {Reunion} et mgt($C$) = {UnderTheSun, PlaceWithBeach, Island}. Les termes de DeSat($C$) et de mgt($C$) apparaissent respectivement encadrés dans les figures fig. 2a et fig. 2b.*

# 3 Méthode de construction de classes

Notre méthode de construction de classes est basée sur la notion de treillis de Galois [10], qui a déjà été utilisée pour faire du raffinement de requêtes [4, 3]. Etant donnés deux ensembles finis $D$ (un ensemble de documents) et $T$ (un ensemble de termes), et une relation binaire $\mathcal{R}$ entre ces deux ensembles, le treillis de Galois est un ensemble particulier de classes, dans lequel chaque classe est un couple, composé d'un sous-ensemble de documents $D' \subseteq D$, appelé **extension** du couple, et un sous-ensemble de termes $T' \subseteq T$, appelé **intention** du couple. Chaque couple $(D', T')$ doit être un couple **complet** pour $\mathcal{R}$, ce qui signifie que $T'$ doit seulement contenir les termes partagés par tous les documents de $D'$, et symétriquement, les documents de $D'$ doivent précisement être ceux qui partagent tous les termes de $T'$.

$T' = f_1(D')$ où $f_1(D') = \{t' \in T' \mid \forall d' \in D', d'\mathcal{R}t'\}$
$D' = f_2(T')$ où $f_2(T') = \{d' \in D' \mid \forall t' \in T', d'\mathcal{R}t'\}$

**Exemple 4** *La figure 3 présente un exemple de 5 documents annotés par 7 termes, {a,b,c, d, e, f, g}. Le document 1 est annoté par les termes {a, c, f}. Le terme g est présent dans les descriptions des documents 2 et 3.*
*$f_1$({1,4}) = {c, f} et $f_2$({c, f}) = {1,4, 5}. ({1,4},{c, f}) n'est donc pas un couple complet (puisque $f_2$({c, f}) $\neq$ {1,4}) alors que le couple ({2, 3},{a, g}) est complet car $f_1$({2, 3}) = {a, g} et $f_2$({a, g}) = {2, 3}.*

Soient deux couples $C_1 = (D_1, T_1)$ et $C_2 = (D_2, T_2)$, une relation d'ordre partiel ($<$) est définie entre les couples par : $C_1 < C_2 \Leftrightarrow T_1 \subseteq T_2$ et $C_1 < C_2 \Leftrightarrow D_2 \subseteq D_1$.

Cet ordre partiel est utilisé pour générer le graphe du treillis de la façon suivante : il existe un arc $(C_1, C_2)$ si $C_1 < C_2$ et s'il n'existe pas d'autre couple $C_3$ dans le treillis tel que $C_1 < C_3 < C_2$. $C_1$

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 1 | • | | • | | | • | |
| 2 | • | | • | | | | • |
| 3 | • | | | • | | | • |
| 4 | | • | • | | | • | |
| 5 | | • | • | | • | • | |

FIG. 3 – *Un exemple de relation entre 5 documents et 7 termes*

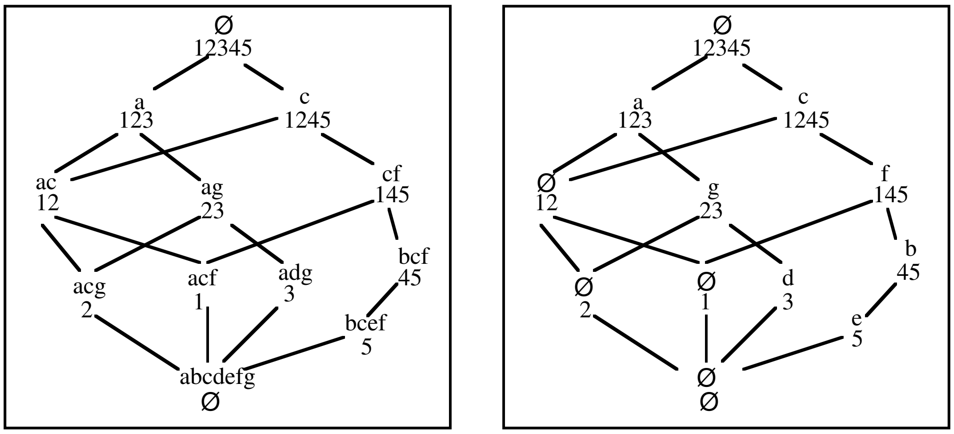


FIG. 4 – *Graphe du treillis de Galois et $T' - ICL$ correspondant*

est dit le **père** de $C_2$ et $C_2$ le **fils** de $C_1$. Le graphe révèle les relations de généralisation/spécialisation entre les couples où $C_1 < C_2$ représente le fait que $C_1$ est plus général que $C_2$.

**Exemple 5** *Dans la figure 4a qui présente le graphe du treillis généré pour la relation de la figure 3, on voit que le couple ({a},{1,2,3}) est plus général que le couple ({a, g},{2,3}). Un ensemble de documents plus grand a moins de termes en commun qu'un ensemble plus restreint.*

Un treillis de Galois est une représentation qui comporte beaucoup de redondances. Pour un couple $C = (D', T')$, $D'$ est présent dans chaque ancêtre de $C$ et réciproquement, $T'$ apparaît dans chacun de ses descendants. Godin et al. ont montré dans [5] que cette redondance peut être éliminée sans perdre d'informations si le graphe est maintenu. Pour un couple $C = (D', T')$, soit $T''$ l'ensemble des termes non redondants de $T'$, $T'' = \{t' \in T' | t' \in f_1(D')$ et tel qu'il n'existe pas d'autre couple $C' = (X, Y) < C$ tel que $t' \in Y\}$.

Godin définit le $T'-$ *inheritance terme lattice* ($T' - ICL$) qui utilise les ensembles de couples $(D', T'')$, et montre que pour un couple donné $C$ la valeur correspondante de $T'$ peut être calculée en faisant l'union des ensembles $T''$ des ancêtres de $C$ y compris $C$ lui-même. Les éléments de $T'$ qui ne sont pas dans $T''$ sont ainsi hérités de ses ancêtres. Dans ce papier, nous utiliserons pour notre algorithme des couples de la forme $(D', T'')$.

**Exemple 6** *En comparant les figures 4a et 4b, on voit que le couple ({a, c},{1,2}) de la figure 4a n'a plus pour intention dans la figure 4b que $\emptyset$. Son intention peut être recalculée en prenant l'union des intentions de ses ancêtres, les couples ({a},{1,2,3}), ({c},{1,2,4,5}) et ($\emptyset$, {1,2,3,4,5}).*

La structure des treillis de Galois est très intéressante dans le contexte de la construction de classes car le contenu d'une classe $(D', T')$ est directement étiqueté par son intention $T'$ qui est un simple ensemble de termes immédiatement lisible et compréhensible par l'utilisateur. De plus, cette structure de treillis autorise le recouvrement des classes : un document peut apparaître dans différentes classes s'il possède différents descripteurs.

Godin[4] et Carpineto[3] ont utilisé ce type de structure pour indexer une base de documents décrits par un ensemble de termes. Une fois construit le treillis entier caractérisant la totalité de la base, les requêtes des utilisateurs qui interrogent la base sont rapprochées des intentions des classes du treillis. En effet, pour chaque classe, son intention peut être vue comme une requête conjonctive et son extension, comme les documents répondant à cette requête. Répondre à une requête de l'utilisateur consiste donc, dans ce contexte, à rechercher la classe la plus générale dont l'intention contient tous les termes de la requête considérée. Une fois cette classe identifiée,[2] l'utilisateur peut naviguer dans le treillis et explorer les classes voisines : les fils de la classe correspondent à un pas de raffinement de la requête et ses parents à un pas d'élargissement.

Dans notre contexte où l'ensemble des documents (des ressources) à classer n'est pas un ensemble déterminé à l'avance mais est généré dynamiquement à partir de la requête de l'utilisateur, la construction du treillis complet serait un processus trop couteux. Pour l'éviter, nous ne construisons que les premiers nœuds du treillis (son premier niveau) et nous identifions dans chaque nœud des axes de spécialisations potentielles de ce nœud. Ainsi l'interaction avec l'utilisateur lui permet de choisir un nœud et un axe et de relancer la construction d'un nouveau niveau du treillis en n'utilisant que les documents apparaissant dans l'extension du nœud sélectionné. Remarquons que comme nous ne construisons donc pas réellement un treillis, et que certains documents ne sont plus pris en compte, nous autorisons l'utilisateur à revenir sur ses choix pour explorer d'autres nœuds.

# 4 Algorithme

Etant donnée une requête de l'utilisateur, les entrées de l'algorithme sont :

• l'ensemble $D$ des identifiants des ressources dont la description s'apparie avec tous les termes de la requête,

• pour chaque identifiant $d_i$ de $D$, un ensemble de termes appelé **description courante** Cdesc($d_i$), qui est initialisé par le saturé de l'ensemble des termes décrivant la ressource.

**Exemple 7** *Supposons que la requête de l'utilisateur soit Q = ResidencePlace. Supposons aussi (pour construire un tout petit exemple) que le moteur de recherche ne trouve que quatre ressources correspondant à cette requête et que les ensembles de termes décrivant ces ressources (ou descripteurs) soient respectivement :*

- $d_1$*: {Hotel, Located, Reunion},*
- $d_2$*: {Hotel, Located, Carribean},*
- $d_3$*: {B&B, Located, Sahara},*
- $d_4$*: {B&B, Located, Corsica}.*

*Après calcul du saturé de l'ensemble des termes décrivant la ressource* $d_1$*, sa description courante est alors initialisée comme suit : Cdesc(*$d_1$*) = {Hotel, ResidencePlace, Located, Reunion, AlwaysSunny, UnderTheSun, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island, GeoPlace, PhysicalPlace}.*

La sortie de l'algorithme est une spécialisation de la requête initiale.

L'algorithme fonctionne comme suit : à chaque itération, (chaque niveau du treillis), il réalise trois tâches :

1- la construction de la classe racine du niveau,

2- la construction des classes du niveau,

3- l'identification des axes d'affinement possibles de chaque classe,

puis il passe la main à l'utilisateur qui choisit une classe, puis décide soit d'itérer le processus en sélectionnant un axe de spécialisation soit de l'arrêter.

2. La classe recherchée peut ne pas exister si la requête posée n'a pas de réponse dans la base. Le problème des réactions coopératives à des réponses vides n'est pas étudié ici et le lecteur intéressé par ce problème pourra se référer à [3], [2] ou [8].

## 4.1 Construction de la classe racine du niveau

Les descripteurs partagés par toutes les ressources de $D$ doivent être identifiés comme étant l'intention de la classe racine du niveau. En effet, les ressources apparaissant dans $D$ sont celles retournées par le moteur de recherche, donc les descriptions courantes de chacune de ces ressources partagent au moins les descripteurs qui vérifient les termes de la requête de l'utilisateur. Dans les itérations suivantes, un certain nombre de descripteurs aussi seront communs à toutes les ressources considérées. Cet ensemble de descripteurs communs est retiré de la description courante de chaque ressource et, une fois désaturé, cet ensemble est conservé comme intention de la classe dont l'extension est l'ensemble de ressources considérées dans le niveau. Ainsi, nous éliminons les termes redondants de $T'$ comme dans les $T'$-$ICL$ de Godin [5].

**Exemple 8** *Les descriptions courantes des quatres ressources de l'exemple partagent les descripteurs suivants : {ResidencePlace, Located, UnderTheSun, GeoPlace, PhysicalPlace}. La classe racine construite avec le désaturé de cet ensemble est (1234, {ResidencePlace, Located, UnderTheSun}) puisque* GeoPlace *et* PhysicalPlace *sont des généralisants de* UnderTheSun.

## 4.2 Construction des classes du niveau

Dans cette étape, nous travaillons sur les descriptions courantes de chaque ressource dont les termes partagés identifiés à l'étape précédente ont été retirés.

Nous commencons par identifier l'ensemble des **termes les plus généraux** du niveau, Mgt, qui est l'union des termes les plus généraux apparaissant dans la description courante de chaque ressource du niveau, Mgt = $\bigcup_{i \in [1 \ldots n]}$ mgt(Cdesc($d_i$)).

**Exemple 9** *Pour la description courante de $d_1$, Cdesc($d_1$) = {Hotel, Reunion, AlwaysSunny, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island}, mgt(Cdesc($d_1$)) = {Hotel, AlwaysSunny, OutOfEurope, PlaceWithBeach, Island}.*

*Pour les quatre ressources considérées, Mgt = {Hotel, B&B, AlwaysSunny, SunnyOnlySummer, OutOfEurope, InEurope, PlaceWithBeach, NoBeachPlace, Island}.*

Puis, pour chaque terme $t$ de Mgt, l'algorithme construit le couple complet $(D', T')$ centré autour de $t$, i.e., tel que les ressources de $D'$ sont celles dont la description courante contient $t$ et tel que $T'$ est l'ensemble des termes communs à toutes les descriptions courantes des ressources de $D'$. Une fois construits les différents couples correspondant à chaque $t$ de Mgt, nous ne conservons que les couples les plus généraux, i.e., ceux dont l'extension n'est pas incluse dans l'extension d'un autre couple complet.

**Exemple 10** *Le couple complet centré autour de AlwaysSunny est : (123, {AlwaysSunny, OutOfEurope}).
Il est plus général que celui centré autour de Hotel, qui est : (12, {Hotel, AlwaysSunny, OutOfEurope, PlaceWithBeach}).
Seul le premier couple est conservé.*

Les couples les plus généraux obtenus ici sont les mêmes que ceux calculés avec un algorithme classique de construction descendante de treillis de Gallois, mais ils sont moins couteux à obtenir.

Un algorithme classique descendant construit les couples complets en calculant la fermeture de chaque terme apparaissant dans une description courante, alors que nous ne calculons que la fermeture de chaque $t \in$ Mgt. Supposons qu'il existe un terme $t$ tel que $t \notin$ Mgt et qu'il existe un couple complet $(X, Y)$ centré autour de $t$ qui soit construit par l'algorithme classique. Si $t \notin$ Mgt cela signifie que dans chaque description où $t$ apparaît il existe un terme $t'$ tel que $t'$ est un généralisant de $t$ et $t' \in$ Mgt. Toute les descriptions des ressources de $X$ contenant $t$, elles contiennent aussi $t'$. Cela signifie que $t' \in Y$ et que le couple complet $(X, Y)$ a aussi été construit par notre algorithme.

L'intention de chacun des couples les plus généraux construits sera présentée à l'utilisateur, lui permettant ainsi de choisir la classe qui lui semblera la plus pertinente.

**Exemple 11** *Trois classes sont finalement conservées à cette étape pour être présentées à l'utilisateur :*
*- (34, {B&B})*
*- (123, {AlwaysSunny(y), OutOfEurope})*
*- (124, {PlaceWithBeach, Island, IslandWithBeach})*

## 4.3 Identification des axes possibles d'affinement des classes

Pour aider l'utilisateur à affiner la classe qu'il choisira, nous voulons lui proposer, associé à chaque classe, un ensemble d'axes possibles d'affinement, c.à.d, un ensemble de termes parmi lesquels il pourra choisir celui qu'il veut voir apparaître dans la future requête. Dans chaque classe, l'ensemble des descripteurs qui seront proposés comme axes sont sélectionnés en recherchant à nouveau les termes les plus généraux parmi les descripteurs des ressources de la classe, une fois retirés bien sûr, les descripteurs apparaissant dans l'intention de la classe. L'identification des axes associés à chaque classe implique donc tout d'abord de mettre à jour, dans chaque classe, les descriptions courantes de chaque ressource.

Dans ce but, dans chaque classe, des copies des descriptions courantes de chaque ressource de la classe sont tout d'abord créées, liées à la classe et mises à jour : les descripteurs apparaissant dans l'intention de la classe sont retirés de la description courante de chaque ressource.

**Exemple 12** *A l'étape précédente, la description courante de $d_1$ était Cdesc($d_1$) = {Hotel, Reunion, AlwaysSunny, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island}. Après copie et mise à jour dans les 2 classes où cette ressource apparaît, ses nouvelles descriptions courantes sont maintenant :*
*- pour la classe (123, {AlwaysSunny, OutOfEurope}),*
*Cdesc($d_1$) = {Hotel, Reunion, IslandWithBeach, PlaceWithBeach, Island}.*
*- pour la classe (124, {PlaceWithBeach, Island, IslandWithBeach}),*
*Cdesc($d_1$) = {Hotel, Reunion, AlwaysSunny, OutOfEurope}).*

Puis, à nouveau, nous recherchons l'ensemble des termes les plus généraux issus de chaque description courante Cdesc($d'_i$), Mgt = $\bigcup_{i \in [1 \ldots n]}$ mgt(Cdesc($d'_i$)), et ce, dans chaque classe.
Le Mgt de chaque classe est présenté, avec celle-ci, comme l'ensemble des axes de spécialisation possibles de la classe. Dans l'affichage à l'utilisateur, chaque axe de spécialisation est suivi par le nombre de ressources de la classe qui le vérifient.

**Exemple 13** *Pour la classe (123, {AlwaysSunny, OutOfEurope}), mgt(Cdesc($d_1$)) = mgt(Cdesc($d_2$)) ={Hotel, PlaceWithBeach, Island} et mgt(Cdesc($d_3$)) = {B&B, NoBeachPlace}.*
*Le Mgt de la classe est {Hotel, B&B, PlaceWithBeach, Island, NoBeachPlace} et l'ensemble des axes affiché est {Hotel 2, B&B 1, PlaceWithBeach 2, Island 2, NoBeachPlace 1}.*

## 4.4 Interaction avec l'utilisateur

A ce stade, le système a affiché les différentes classes et leurs axes de spécialisation possibles. L'utilisateur doit choisir une classe $(D', T')$, puis il peut soit :

• choisir un axe de spécialisation $t$ parmi les termes proposés avec la classe sélectionnée. L'algorithme identifie alors le sous-ensemble de ressources de $D'$ dont la description courante contient ce terme $t$, et réitère l'algorithme en reprenant à la tâche 1, avec ce sous-ensemble de ressources.

**Exemple 14** *Supposons que l'utilisateur choisisse la classe (123, {AlwaysSunny, OutOfEurope}) et le terme* PlaceWithBeach *comme axe d'affinement. Parmi les 3 ressources de la classe, seules les ressources 1 et 2 vérifient ce terme, et c'est sur elles et leur description courante actuelle que l'algorithme reprend à l'étape 1.*

• signifier que la classe sélectionnée est satisfaisante. La requête affinée correspondant à cette classe est alors construite en faisant l'union des intentions des classes successivement choisies par l'utilisateur, puis en calculant le désaturé de l'ensemble de descripteurs ainsi constitué.

**Exemple 15** *Supposons ici aussi que l'utilisateur choisisse la classe (123, {AlwaysSunny, OutOfEurope}) mais qu'il dise que cela lui convient.*
*La requête affinée est composée de l'union des ensembles suivants : {ResidencePlace, Located, UnderTheSun} qui est l'intention de la classe racine, calculée dans l'exemple 8 et {AlwaysSunny, OutOfEurope} qui est celle de la classe qui vient d'être choisie. Après désaturation, la requête résultante est {ResidencePlace, Located, AlwaysSunny, OutOfEurope}.*

Bien évidemment si les classes construites lui déplaisent, l'utilisateur peut toujours remonter à un choix précédent dans l'arbre construit et explorer une autre branche. Le mécanisme proposé lui permet ainsi d'effectuer pas à pas les étapes d'affinement comme il aurait pu le faire en naviguant dans un vrai treillis de Galois : en effet, pour une classe, chaque mgt proposé comme axe de spécialisation correspond à une des caractéristiques que présenteraient certains des fils de cette classe dans le vrai treillis de Galois. Mais le temps de construction de notre arbre est moindre ou du moins fractionné si l'utilisateur veut finalement explorer toutes les branches de l'arbre.

# 5 Travaux proches et conclusion

De nombreux travaux se sont déjà attaqué au problème de l'organisation des documents retournés par un moteur de recherche en réponse à une requête. Pour éviter d'imposer à l'utilisateur de parcourir une longue liste de documents classés selon leur pertinence supposée, certains chercheurs ont déjà proposé de regrouper ces documents en classes cohérentes par des techniques dites de **classifications éphémères** [7, 11]. Les exigences de ce type de classification ont été définies par Maarek et al.[7] : il nécessite d'une part une grande précision car l'utilisateur qui n'est la plupart du temps pas un expert d'un domaine est peu tolérant aux erreurs et d'autre part une forme de présentation qui lui permette de parcourir aisément les classes, avec des techniques de visualisation et/ou d'étiquettage automatique des classes construites.

Pour augmenter la précision, Maarek et al. utilisent une méthode de Construction Hiérarchique par Agglomération (HAC) et définissent une mesure de similarité entre documents basée sur une unité d'indexation qui consiste en des paires de mots reliés par affinité lexicale. Pour améliorer l'étiquettage automatique des classes construites et pour en présenter une description concise et précise, Zamir et Etzioni [11] effectuent leur classification en construisant un arbre de suffixes extraits des phrases issues des résumés associés aux documents retournés par le moteur de recherche. Mais ces travaux n'acceptent pas les réactions de l'utilisateur et ne l'aident pas à reformuler sa requête initiale.

Pour augmenter la recherche interactive et prendre en compte les retours de l'utilisateur Leuski et Allan [6] combinent liste ordonnée et construction de classes : les documents sont ordonnés suivant leur probable pertinence et l'utilisateur est supposé suivre cet ordre jusqu'à ce qu'il trouve le premier document réellement intéressant. Ce premier document sert alors à initialiser la construction de classes avec les documents non encore examinés.

Berenci et al. aident l'utilisateur à se focaliser sur les termes pertinents et à reformuler sa requête en visualisant les documents retournés regroupés par paquets suivants les différents sous-ensembles de termes de la requête initiale qu'ils vérifient [1].

Les travaux basés sur les treillis de Galois [4, 3] présentés en section 3 sont les plus proches des notres. En effet cette méthode ne nécessite ni l'utilisation de mesure de similarité comme [7, 6] ni l'analyse de phrases complètes comme [11]. Ses avantages sont de deux sortes. D'une part, l'étiquettage d'une classe est fait automatiquement : c'est l'intention de la classe et elle recouvre l'ensemble des termes partagés par tous les documents de la classe. D'autre part, le treillis permet d'explorer facilement les spécialisations d'une requête en explorant les voisins de la classe représentant la requête dans le treillis. Le principal problème de cette méthode de construction de classe est son coût : elle ne peut être appliquée en temps réel sur de trop gros ensemble de documents.

Dans notre contexte d'entrepôt thématique, le nombre de documents retournés, même s'il est trop important pour être géré facilement par l'utilisateur, reste raisonnable. De plus, notre algorithme ne nécessite pas de construire le treillis dans sa totalité : cette méthode de construction de classes retrouve ainsi toutes ses qualités. Un des aspects importants de notre travail est l'utilisation d'une ontologie du domaine dans laquelle chaque terme a de multiples généralisants et spécialisations. L'usage de cette ontologie permet d'enrichir la description d'une ressource par de multiples points de vue qui sont utilisés dans la phase de construction des classes et augmentent ainsi sa précision. L'algorithme présenté ici est implémenté en Java et actuellement expérimenté dans le médiateur PICSEL [9] dans le cadre d'un contrat avec France-Télécom R&D.

# Références


[1] Ezio Berenci, Claudio Carpineto and Vittorio Giannini. Improving The Effectiveness Of Web Search Engines Using Selectable Views Of Retrieval Results. *Journal of Universal Computer Science*, 4, 737-747, 1998.

[2] A. Bidault, Ch. Froidevaux and B. Safar, Repairing Queries in a Mediator approach. In *Proc. of ECAI 2000*, pp. 406-410, Berlin, August 2000.

[3] Claudio Carpineto and Giovani Romano. Information Retrieval through hybrid navigation of lattice representations. *International Journal of Human-Computer Studies*, 45, 553-578, 1996.

[4] Robert Godin, Rokia Missaoui and Alain April. Experimental comparison of navigation in a Gallois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38, 747-767, 1993.

[5] Robert Godin, Rokia Missaoui and H. Alaoui. Incremental Concept Formation Algotithms Based on Galois (Concept) Lattices. *Computational Intelligence,* 11(2), 246-267, 1995.

[6] Anton Leuski and James Allan. Improving Interactive Retrieval by Combining Ranked Lists and Clustering. In *Proceedings of RIAO'2000*. 665-681, April 2000.

[7] Yoelle Maarek, Ronald Fagin, Israel Ben-Shaul, and Dan Pelleg. Ephemeral Document Clustering for Web Applications. IBM Research Report RJ 10186, April 2000.

[8] T. Gaasterland, P. Godfrey, and J. Minker. An overview of Cooperative Answering. *Journal of Intelligent Information Systems*, vol.1, pp. 123-157, 1992.

[9] M.-Ch. Rousset, A. Bidault, Ch. Froidevaux, H. Gagliardi, F. Goasdoué, C. Reynaud et B. Safar, Construction de médiateurs pour intégrer des sources d'information multiples et hétérogènes : le projet PICSEL, Dans *Revue I3,* Vol 2,(1), pp. 9-59, Septembre 2002.

[10] R. Wille. Concept Lattices and Conceptual Knowledge Systems. *Computers and Mathematics with Applications* 23, 403-515, 1992.

[11] Oren Zamir and Oren Etzioni. Web document Clustering: A Feasibility Demonstration. In *Proceedings of SIGIR'98*, 46-54, Melbourne, August 1998.