Recherche de services dans les grilles de calcul : approche par les treillis de Galois

Mohamed Lassoued

Faculté des Sciences de Tunis, 1060 Tunis, Tunisie

mohamed.lassoued@gmail.com

Samir Moalla

Faculté des Sciences de Tunis, 1060 Tunis, Tunisie

samir.moalla@fst.rnu.tn

Résumé

Les grilles de calcul constituent de nos jours des infrastructures pouvant contenir un large éventail de ressources et de services. Avec le nombre croissant de services, les utilisateurs ne peuvent pas les manipuler facilement avec les méthodes traditionnelles. Ainsi, ont-ils besoin d'un mécanisme qui puisse les assister pour la gestion des services dans les grilles. Les méthodes traditionnelles, fondées sur la classification et la recherche par mots-clés ou par catégories présentent beaucoup d'insuffisances. Cet article propose une approche formelle basée sur les treillis de Galois pour la gestion des services dans les grilles de calcul.

Abstract

The grid in now an infrastructure containing a broad range of resources and services. With the number growing of services, the users cannot easily handle them with the traditional methods. Thus, they need a mechanism which can assist them for the management of the services in the Grid. The traditional methods, based on classification and the keyword search or by categories present much insufficiencies. This article proposes a formal approach based on the Galois lattices for the management of the services in the Grids.

Mots-clés

grille, service Web, gestion de service, recherche, treillis de Galois, revue eTI, revue électronique, libre accès

Keywords

grid, Web service, service management, find a service, Galois lattice, eTI, electronic journal, open access

1. Introduction

Les grilles de calcul représentent une infrastructure capable de fournir des capacités de calcul, de stockage et de communication d'une manière transparente à l'utilisateur. Par définition, un service de grille est un service Web particulier qui fournit des interfaces bien définies et qui suit des conventions spécifiques (Al-Ali, Rana *et al.*, 2004), (Foster, 2001). Un service Web est un composant logiciel caractérisé par son identifiant, noté URI, dont les interfaces publiques sont définies et appelées dans un document XML. Il peut être découvert par d'autres systèmes logiciels, ou d'autres services Web, par l'intermédiaire d'un dépôt central (UDDI) (Benatallah, Hacid *et al.*, 2005), (Oberle, Lamparter *et al.*, 2005). L'avantage principal d'un service Web est la capacité de créer des applications par l'utilisation de composants logiciels faiblement connectés et réutilisables (Chinniciet, Moreau *et al.*, 2006), (Tjoa, Brezany *et al.*, 2003).

Les services Web présentent un grand intérêt dans de nombreux domaines, entre autres dans les systèmes à grande échelle. De nombreux services ont été proposés pour permettre aux utilisateurs de développer et de déployer des applications à large échelle (Fox, Pallickaran *et al.*, 2003), (Oberle, Lamparter *et al.*, 2005). Afin de permettre une découverte efficace de ces services, nous avons besoin d'une plateforme performante, facile à utiliser et ouverte pour intégrer un grand nombre de services. Une telle plateforme devrait rendre possible la gestion efficace de services et en particulier, leur recherche à travers une grille. En effet, l'opération de recherche est essentielle puisqu'elle est nécessaire dans la plupart des autres opérations appliquées sur les services comme l'ajout, la modification, la suppression, la composition, *etc.*

Actuellement, chaque système de grille a un répertoire central dans lequel les services sont classés manuellement dans diverses catégories. Dans le système Globus par exemple, le répertoire principal est le *Metacomputing Directory Service* (MDS) (Denayer, 2004). Ce répertoire fournit un accès à l'information statique et dynamique concernant les nœuds de la grille et ses ressources. Il permet aux utilisateurs et aux autres applications de découvrir les données concernant les services et de les superviser. Si un utilisateur cherche un service donné, il passe en revue manuellement ce répertoire par catégorie ou par mot-clé (Denayer, 2004), (Shi, Zhang *et al.*, 2005). Cette recherche qui n'est pas formelle, est fondée sur des règles ou des normes fixées par les fournisseurs de services. Si le demandeur n'est pas informé de l'emplacement du service ou de ses caractéristiques, il pourra difficilement trouver le service demandé. En raison de cette limitation, l'efficacité et l'exactitude de la recherche d'un service ne peuvent pas être garanties (Potts, Sedukhin *et al.*, 2003), (Wang, Huang *et al.*, 2003). Une bonne solution serait d'utiliser une méthode formelle pour modéliser la représentation de services dans les répertoires (Bruno, Canfora *et al.*, 2005), ce qui faciliterait leur recherche dans une grille.

Le reste de cet article est organisé comme suit. La section 2 passe en revue les travaux liés à la recherche de services. La section 3 présente un modèle formel pour la recherche de services Web. Un algorithme de recherche et un exemple d'exécution sont décrits dans la section 4. Enfin la section 5 conclut ce papier et dégage quelques perspectives de travaux futurs.

2. Recherche de services dans une grille

Parfois, dans les répertoires d'une grille ou de services Web, il est difficile de déterminer avec exactitude à quelle catégorie un service pourrait appartenir, particulièrement dans le cas de ceux qui exécutent plusieurs fonctionnalités (Bruno, Canfora *et al.*, 2005), (Denayer, 2004). Par exemple, supposons qu'il existe dans le répertoire d'une grille trois catégories de services : ***Management***, ***Reservation*** *et* ***Execution***. Un fournisseur de services veut publier un nouveau service de grille appelé ***LigthExecution*** effectuant les trois opérations suivantes :

- *SearchIfExist* qui indique si un service est en exécution,
- *AllocateNode* qui réserve un nœud spécifique de la grille,
- *UpdateInstance* qui permet de mettre à jour une instance d'un service.

La question est : dans quelle catégorie ce service doit-il être classé ?
Ce choix dépend du fournisseur de service. Mais, quelque soit l'endroit où le service sera placé, ceci ne garantit pas la localisation de ce service lors d'une recherche. Par exemple, si le service est classé dans la catégorie ***Reservation*** (puisqu'il offre une opération de réservation), un utilisateur qui cherche un service fournissant une opération de gestion d'un nœud, ne pourra pas le trouver. Au contraire, si le service est affecté à la catégorie ***Management***, l'utilisateur qui cherche un service ayant la capacité de réservation, ne pourra pas également le trouver. Une méthode intuitive est d'enregistrer ce service dans toutes les catégories, ce qui peut générer de la redondance dans le répertoire.
Notre modèle est fondé sur une recherche orientée opération qui permet à un utilisateur de chercher un service (ou un ensemble de services) exécutant un ensemble d'opérations.

2.1. Caractéristiques de recherche

La recherche d'un service est une action importante dans la gestion de services. En effet, toute utilisation d'un service nécessite d'abord sa recherche. Par exemple, pour la suppression d'un service, nous devons d'abord le rechercher avant de le supprimer.
Dans le cas général, un mécanisme de recherche doit avoir les propriétés suivantes (Denayer, 2004) :

- Précis *i.e.* les résultats retournés doivent correspondre aux critères de recherche.
- Rapide *i.e.* le temps de réponse doit être le plus court possible.
- Efficace *i.e.* la totalité des résultats ne devrait pas être trop grande. Le fait de donner un résultat ayant une cardinalité faible et un degré de correspondance élevé est plus intéressant que de produire des résultats importants avec des degrés de correspondance faibles.

La particularité de l'opération de recherche dans le domaine des services réside dans la notion même de service. En effet, la réponse à la requête d'un utilisateur n'est pas booléenne. Dans le cas de recherche d'un service, plusieurs réponses sont possibles :

- aucun service ne couvre les opérations demandées ;
- un ou plusieurs services couvrent exactement les opérations demandées ;
- un ou plusieurs services couvrent totalement les opérations demandées mais offrent d'autres opérations ;
- aucun service ne couvre totalement les opérations demandées et l'union des services trouvés ne couvre pas les opérations demandées ;
- aucun service ne couvre totalement les opérations demandées mais l'union des services trouvés couvre les opérations demandées.

2.2. Etat de l'art de la recherche de services

Benatallah, Hacid *et al.* (2005) utilisent les logiques de description (DL's) comme cadre formel pour la gestion automatique de services. Les DL's sont une famille de logiques qui ont été développées pour modéliser les structures hiérarchiques complexes et les doter d'un moteur de raisonnement spécialisé (Benatallah, Hacid *et al.*, 2005).
D'autres travaux présentant la notion de service Web sémantique ont été proposés dans (Fox, Pallickaran *et al.*, 2003), (Merwe, Obiedkov *et al.*, 2004). Ces travaux s'orientent vers la création de normes sémantiques pour permettre l'automatisation complète de la gestion (et la recherche) de services Web. Ils ne sont pas fondés simplement sur la syntaxe du service, mais également sur sa sémantique. (Fox, Pallickaran *et al.*, 2003) et utilisent la notion de similitude pour rechercher des services. Cependant, il peut être peu convenable de chercher

un service selon l'approche de similarité, parce que deux services peuvent ne pas être semblables globalement mais ils peuvent l'être au niveau des opérations.
D'autres travaux utilisent l'ontologie de service dont les instances représentent les services Web (Kovacs, 2003). L'idée est de pouvoir faire la recherche sur le contenu du service plutôt que sur la base de mots-clés. L'ontologie est utile dans le contexte des services de grille car elle permet d'unifier ou de normaliser les caractéristiques des services demandés et fournis.
L'ensemble de ces travaux ne satisfait pas, dans tous les cas, les exigences des utilisateurs. En effet, les utilisateurs recherchent un service doté d'opérations spécifiques car, pour un utilisateur, les opérations exécutées par un service sont plus importantes que les autres facteurs tels que son nom, sa localisation, sa catégorie, *etc.*

3. Approche proposée

Notre approche privilégie une recherche fondée sur les opérations exécutées par un service. En fait, quand un service est publié, son fournisseur précise dans un fichier WSDL (*Web Service Description Language*), la description de ce service (Chinniciet, Moreau *et al.*, 2006). Ce fichier décrit le type du service Web (opérations réalisées et types des paramètres) et les aspects techniques d'implantation du service. Notre travail exploite ce fichier qui contient plusieurs parties. La partie appelée *PortType* décrit les opérations offertes par ce service. Dans la suite, nous désignons par opération d'un service, une opération décrite dans la partie *PortType*.

3.1. Définitions de base

Dans cette section, nous introduisons les treillis de Galois que nous utiliserons pour la définition de notre approche.
Un contexte formel est un riplet $(\mathcal{S}, \mathcal{O}, \mathcal{R})$ où S est un ensemble de services, O un ensemble d'opérations, et R une relation binaire $\subseteq \mathcal{S}x\mathcal{O}$. Soit $s \in \mathcal{S}$ et $o \in \mathcal{O}$, la notation *sRo* indique que le service **s** exécute l'opération **o**. Cette relation peut être représentée par une table T, appelée table de contexte, où T[i,j]=1 si S_iRo_j et 0 sinon (voir Table 1) (Berry, McConnell *et al.*, 2006), (Bruno, Canfora *et al.*, 2005), (Hao et YoungQiang, 2004).

	Opérations				
Services	Op1	Op2	Op3	Op4	Op5
s1	1	0	0	1	0
s2	0	0	1	1	0
s3	1	1	0	0	0
s4	1	0	1	0	1
s6	0	1	1	0	1

Table 1. Exemple de table de contexte

Un service exécute au moins une opération et n'importe quelle opération est exécutée, au moins, par un service.
Ayant un sous-ensemble d'opérations $O = \{op_1,op_2,...,op_k\}$ et un sous-ensemble de services $S=\{s_1,s_2,...,s_l\}$, un concept est un couple (S,O) où f(S)=O, et g(O)=S (Berry, McConnell *et al.*, 2006), (Hao et YoungQiang, 2004), (Ludwig et Santen, 2002).
S est appelé ***intend*** de O et représente le sous-ensemble des services qui réalisent toutes les opérations de O. O est appelée *l'**extend*** de S et contient toutes les opérations communes à tous

les services de l'ensemble S.
L'ensemble des concepts peut être représenté par un graphe dont les nœuds sont les concepts et les arcs sont les relations entre concepts. Ce graphe (ou diagramme) peut être classé par niveaux. Au niveau le plus haut (niveau 0), nous trouvons le concept dont *l'**extend*** contient toutes les opérations et *l'**intend*** est l'ensemble vide. Le niveau 1 représente les concepts ayant en commun une seule opération. Au niveau **i**, figurent tous les concepts dont le nombre d'opérations communes est égal à **i**. Le dernier niveau rassemble tous les services en ***intend*** et l'ensemble vide en ***extend***.

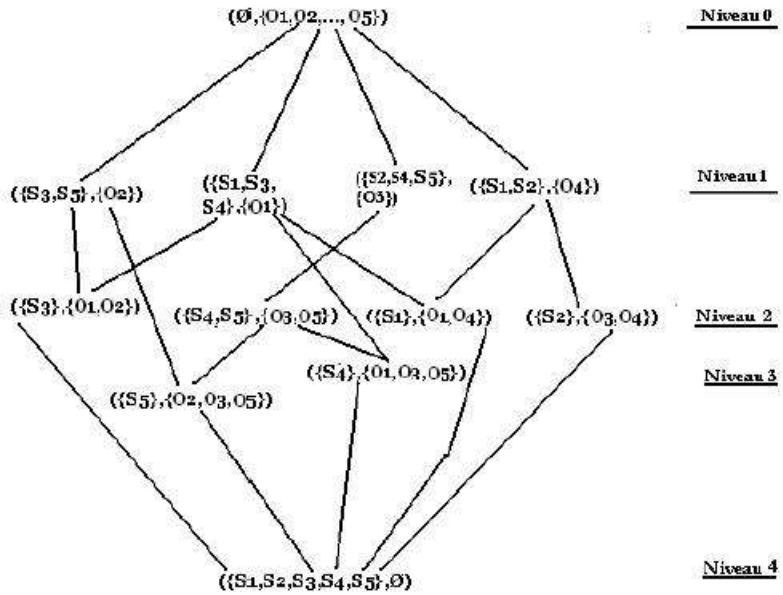

Figure 1. Graphe de représentation

En parcourant ce diagramme de haut en bas, nous avons de plus en plus d'opérations communes dans les concepts (plus spécifiques), et des ***intend*** (services) plus généraux. Nous proposons d'utiliser cette hiérarchie (opération, service) pour la recherche de services fondée sur les opérations. Ainsi, est-il important de classer les concepts avec le minimum d'opérations communes au niveau le plus haut du graphe.
L'exemple de la table 1 comprend cinq services et cinq opérations. Le diagramme qui lui est associé est illustré par la figure 1. Ce diagramme contient plusieurs concepts : quatre d'entre eux relèvent du niveau 1 avec une seule opération dans *l'**extend*** tandis que quatre du niveau 2 ont chacun deux opérations dans l'***extend***, *etc.*

3.2. Algorithme de recherche

Nous supposons qu'un utilisateur recherche un service qui exécute des opérations formulées dans une requête.
Notre algorithme de recherche parcourt les concepts de haut en bas (selon le graphe) pour tenter de répondre à la requête de l'utilisateur. Ainsi, si l'algorithme trouve un service qui peut satisfaire la demande de l'utilisateur, il s'arrête. Le comportement de l'algorithme proposé peut se définir comme suit :

- l'étape 1 calcule les niveaux et affecte des concepts à ces niveaux (lignes 1-6) ;
- l'étape 2 initialise les variables (lignes 7-9) ;
- l'étape 3 parcourt un seul niveau pour chercher les opérations de la requête utilisateur (lignes 11-16) ;
- l' étape 4 parcourt tous les niveaux (lignes 10-18).

Algorithme

Input :
C = (W, O, R) un contexte.
Req = {Op_1,Op_2,...,Op_k} une requête.
Output :
Un ensemble de services qui réalisent les opérations de la requête.

```
GraphLevel = CalculateLevels()
Level[][] = ∅
For i = 1 to ConceptsNum Do
        j = Level(Concept(W,O))
                Level[j][] = Level[j][] + Concept(W,O)
End For
Res = ∅
Found = False
i = SizeOf(Req)
Repeat
        For j=1 to Levelsize[i] do
                The current concept is (W,O)  /* W set of services
                If Req ⊆ O Then       /* O set of operations
                                Res = Res ∪ {W-Res}
                                Found =True
        End For
        i=i+1
Until i>=GraphLevel or Found = True
Return Res
```

Dans la première partie de l'algorithme, nous commençons par le calcul du niveau du graphe et le niveau de chaque concept (lignes 1-6) et nous stockons chaque concept dans son niveau. Le niveau d'un concept est le nombre d'opérations dans l'***extend*** du concept. La valeur initiale du résultat est l'ensemble vide noté $\emptyset$ (ligne 7). Nous commençons le parcours par le niveau qui est égal au nombre d'opérations dans la requête (lignes 9-10). Pour chaque niveau, l'algorithme passe en revue les concepts un par un (ligne 11) et teste si la requête est égale ou est incluse dans le sous-ensemble O, *l'**extend*** du concept (lignes 12-13). Si le test est concluant, alors il faut ajouter *l'**intend*** du concept, *i.e.* les services Web, au résultat (ligne 14). Pour éviter la répétition d'un service dans le résultat, nous vérifions s'il est déjà apparu dans ce résultat (ligne 14). Si nous trouvons un résultat pour le niveau actuel, l'algorithme s'arrête et nous retournons le résultat. Dans l'autre cas, nous irons au niveau suivant (ligne 17) et répéterons le même travail au niveau suivant (ligne 18). Cette répétition se termine si nous trouvons un résultat ou si tous les niveaux sont parcourus sans résultat (ligne 18). Il est important de noter que la première partie de l'algorithme (lignes 1-6) sera exécutée une seule fois. Nous ne devons pas recalculer les niveaux pour chaque opération de recherche, à moins que le graphe ait subi des modifications (nouveau service, suppression ou modification d'un service, *etc*.).
Soit H le nombre de niveaux dans un graphe et L le nombre maximum de nœuds d'un niveau. Les complexités minimales et maximales de notre algorithme sont respectivement de O(L) et O(H*L). Dans le premier cas ceci veut dire que nous avons trouvé le service au premier niveau parcouru. Dans le second cas, l'algorithme parcourt tous les niveaux sans succès. Dans les deux cas la complexité est linéaire.

3.3. Exemples d'application

Dans cette section, nous présenterons deux exemples d'application de l'algorithme proposé. Considérons le contexte donné par la table 2. Ce contexte contient huit opérations et dix services. Dans le premier exemple, nous considérons la requête Req1 = {Op1, Op3, Op7} alors que dans le second, il s'agit de la requête Req2 = {Op1, Op6}.

	Op1	Op2	Op3	Op4	Op5	Op6	Op7	Op8
S1	1	0	1	0	1	0	1	0
S2	1	0	0	1	0	0	1	0
S3	1	0	0	0	1	0	0	1
S4	0	1	0	1	0	0	1	0
S6	0	0	0	0	1	1	0	1
S6	1	1	1	0	0	0	1	0
S7	0	0	1	0	1	0	1	0
S8	0	0	1	0	1	1	0	1
S9	1	1	0	0	0	0	0	0
S10	1	0	0	1	1	0	0	0

Table 2. Exemple de contexte

La recherche de concepts et la création des niveaux du contexte de l'exemple permet de dégager les concepts classés par niveaux, comme indiqué dans la table 3.

Niveau 1	({s4, s6, s9},{Op2}) - ({s1,s2,s4,s6},{Op7}) - ({s2,s4,s10},{Op4}) - ({s1,s2,s3,s6,s9,s10},{Op1}) - ({s1,S6,s8},{Op3}) - ({s1,s3,s6,s7,s8,s10},{Op5})
Niveau 2	({s4, s6},{Op2,Op7}) - ({s2,s4},{Op4,Op7}) - ({s6,s9},{Op1Op2}) -({s1,s2,s6},{Op1,Op7}) - ({s2,s10},{Op1Op4}) - ({s1,s3,s10},{Op1,Op5}) - ({s1,s7,s8},{Op3,Op5}) - ({s3,s6,s8},{Op5,Op8})
Niveau 3	({s4}, {Op2Op4,Op7})-({s1,s6},{Op1,Op3,Op7})- ({s2},{Op1,Op4,Op7}) - ({s10},{Op1,Op4,Op5}) - ({s3},{Op1,Op5,Op8}) - ({s6,s8},{Op5,Op6,Op8})
Niveau 4	({s6},{Op1,Op2,Op3,Op7}) - ({s1},{Op1 ,Op3,Op5,Op7}) - ({s8},{Op3,Op5,Op6,Op8})

Table 3. Niveaux et concepts

Instructions	Résultat d'exécution
Etape 1 : Création des niveaux (lignes 1-6)	La table de niveaux contient les concepts et leurs niveaux respectifs.
Etape 2 : La requête contient 3 opérations Etape 3 : Parcourir le niveau 3 et tester s'il y a un concept dont l'*extend* contient ou est égal à la requête Req1	Commencer la recherche avec le niveau 3 (ligne 9) Le 2^{eme} concept vérifie le test (ligne 13), donc Res={s1, S6} et Found=True
Etape 4 : Test de la boucle (Found =True) (ligne 17)	Arrêter la boucle
Etape 5 : Fin d'exécution	Res est égal à {s1, S6}

Table 4. Table d'exécution pour la requête Req1= {Op1, Op3,Op7}

Dans la table 2, nous remarquons que les seuls services qui fournissent les opérations {Op1, Op3, Op7} sont S1 et S6. On voudrait vérifier si notre algorithme permet d'obtenir ce résultat. La table 4 montre le déroulement de notre algorithme pour le premier exemple et confirme le résultat attendu, à savoir les services S1 et S6.

Instructions	Résultat d'exécution
Etape 1: Création de niveaux (lignes 1-6) Etape 2: La requête contient 2 opérations	Etape déjà exécutée dans l'exemple 1 Commencer avec le niveau 2 (ligne 9)
Etape 3 : Parcourir le niveau 2 et tester s'il y a un concept qui contient ou est égal à {Op1, Op6} (lignes 11-16)	Les concepts du niveau 2 sont différents donc pas de changement des variables (Res, Found)
Etape 4 : Passage au niveau suivant (3) (lignes 17-18) Etape 5 : Parcourir le niveau 3 et tester s'il y a un concept qui contient ou est égal à {Op1,Op6} (lignes 11-16)	Répéter l'étape précédente Les concepts du niveau 3 ne contiennent pas ces opérations donc pas de changement des variables Res, Found
Etape 6: Passage au niveau suivant (4) (lignes 16-17) Etape 7: Parcourir le niveau 4 et faire le test usuel avec l'ensemble {Op1, Op6) (lignes 11-16)	Recommencer la boucle Aucun concept du niveau 4 ne satisfait la requête, donc pas de changement des variables Res et Found
Etape 8: Fin de la boucle (ligne 18)	Arrêter la boucle. Le résultat est toujours $\emptyset$

Table 5. Table d'exécution pour la requête Req2 = {Op1, Op6}

Considérons maintenant la deuxième requête Req2={Op1, Op6}. L'algorithme commence avec le niveau 2 qui correspond au nombre d'opérations dans la requête. Ce niveau ne contient aucun concept dont l'**extend** est égal à la requête. Ensuite, l'algorithme parcourt les niveaux suivants sans succès. Les détails de l'exécution de l'algorithme pour cet exemple sont représentés par la Table 5. Nous remarquons que l'algorithme donne le résultat escompté, à savoir aucun service relatif à la requête.

4. Conclusion

Dans cet article, nous avons proposé une nouvelle méthode pour la recherche de services dans une grille. C'est une méthode formelle de recherche de services basée sur les treillis de Galois. Partant de cette méthode, nous avons développé un algorithme de recherche que nous avons illustré à travers deux exemples.
Comme perspective à court terme, nous pensons définir la notion de similarité entre services pour répondre aux requêtes utilisateurs. Par ailleurs, nous envisageons d'exploiter les potentialités des ontologies dans la recherche de services similaires dans les grilles.

Al-Ali, J., Rana, F., Walker, D., Jha, S., and Sohail, S. (2004). G-qosm : Grid service discovery using qos properties. in *Concurrency and Computation : Practice and Experience Journal, 5 :16—21*.
Benatallah, B., Hacid, M.S., Leger, A., Rey, C., et Toumani, F. (2005). On automating web services discovery. in *The VLDB Journal, 14(1) :84--96, Mars*.
Berry, A., McConnell, R.M., Sigayret, A., et Spinrad. J., (2006). Very fast instances for concept generation. In *Formal Concept Analysis, 4th International Conference*, ICFCA 2006, pages 119-129.
Bruno, M. , Canfora, G. , Penta, M.D., et Sconomiglio. R. (2005), An approach to support web service classiffication and annotation. EEE'05. in *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service one-Technology, e-Commerce and e-Service*, 14(2-3) :138-143.
Chinnici, R. , Moreau, J., Ryman, A. , et Weerawarana, S., (2006) Web services description language (wsdl) version 2.0 part 1 : Core language. W3C candidate recommendation, in *W3C, http ://www.w3.org/TR/wsdl20/*.
Denayer, M.L (2004). Découverte de services : Etude d'approche syntaxique et sémantique. in *Technical report, Bioinformatics Grid Ressources and Environments*, University of Namur, Belgique.
Foster, I. (2001). The anatomy of the grid : Enabling scalable virtual organizations. In IEEE Computer Society, editor, *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, volume 01, pages 6--7, Brisbane, Australia, Mai.
Fox, G., Pallickaran S., Pierce, M., et Walker, D. (2003) Towards dependable grid and web services. in *Ubiquity*, 4(25) :3-9.
Hao, S., et YoungQiang, S., (2004) Web semantics, grid and coalgebra. In IEEE Computer Society, editor, *SCC '04 : Proceedings of the 2004 IEEE International Conference on Services Computing*, volume 01, pages 479--482, Los Alamitos, CA, USA, Avril.
Kovacs, G.L., (2003) Concept lattice structure with attribute lattice. In IEEE Joint Chapter of Robotics, Automation, and Industrial Electronics Society, editors, *4th International Symposium of Hungarian Researchers on Computational Intelligence* 2003, pages 100{115, Budapest, Hungary, Novembre.
Ludwig, S.A. et Santen, P.V. (2002), Position paper : A grid service discovery matchmaker based on ontology description. In *EuroWeb, Electronic Workshops in Computingm*, Decembre.
Merwe, D.V., Obiedkov, S.A., et Kourie, D.G. (2004) Addintent : A new incremental algorithm for constructing concept lattices. In Springer, editor, *Second International Conference on Formal Concept Analysis 2004*, pages 372--385, Sydney, Australia, Février 2004.
Oberle, D., Lamparter, S., Eberhart, A., et Staab, S. (2005) Semantic management of web services. In IEEE Computer Society, editor, ICWS '05 : *Proceedings of the IEEE International Conference on Web Services*, pages 514--519, Juillet.
Potts, M., Sedukhin, I., Kreger, H., et Stokes, E, (2003) Web service manageability specification (WS-Manageability). in *OASIS*, Septembre.
Shi, Y., Zhang, L., Liu, B., Liu, F., Lin, L., et Shi, B. (2005). A formal specification for web services composition and verification. In IEEE Computer Society, editor, *CIT '05 : Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 252--256.
Tjoa, A.M., Brezany, P., et Janciak, I. (2003) Towards grid based intelligent information systems. In IEEE Computer Society, editor, *International Conference on Parallel Processing(ICPP'03)*, Kaohsiung, Taiwan, pages 503--505.
Wang, H., Huang, J.Z., Qu, Y., et Xie, J. (2004) Web services : problems and future directions. Journal of Web Semantics, 1(3) :309--320.
Wang, Y., et Stroulia, E., (2003). Flexible interface matching for webservice discovery. In IEEE Computer Society, editor, WISE'03 : Proceedings of the Fourth International Conference on Web Information Systems Engineering, volume 01, pages 147--156, Décembre.