Systèmes Distribués de Pilotage de Programmes : Application à l'Imagerie Médicale

Naoufel Khayati*[1],** — Wided Lejouad Chaari*[2] — Sabine Moisan**

** ENSI Tunis, Campus Universitaire La Manouba, Tunisie.*
[1]Ur. SOIE, ISG Tunis / [2]GRIFT, ENSI
*** INRIA Sophia Antipolis – Projet Orion, France.*

{naoufel.khayati, sabine.moisan}@sophia.inria.fr ; wided.chaari@ensi.rnu.tn

Résumé

Nous nous intéressons aux Systèmes à Base de Connaissances (SBC) en pilotage automatique de programmes dont le but est d'automatiser l'utilisation de logiciels complexes, grâce à des systèmes capables de planifier des traitements et d'en contrôler l'exécution. Leur distribution devient indispensable car les applications en grandeur réelle impliquent de plus en plus de participants inter ou intra-entreprises sur divers sites. La distribution permet soit la simple consultation distante de bases de connaissances, soit la construction collaborative de bases de connaissances, soit le lancement d'une requête sur un SBC distant, avec des données locales. L'application actuelle porte sur l'assistance de différents médecins en Imagerie médicale et plus précisément pour la détection de l'ostéoporose dans des radiographies d'os de leurs patients. L'assistance vient du fait que le traitement d'une requête sur des images données, par plusieurs programmes distants est transparent aux médecins. Dans cet article, nous proposons une architecture distribuée d'un système de pilotage de programmes illustrée par un scénario pour la résolution des requêtes de médecins.

1. Introduction

De nombreuses bibliothèques de programmes ont été développées par des spécialistes dans différents domaines pour informatiser certains traitements et répondre à des besoins d'une manière automatique. Mais, souvent l'utilisateur final de ces bibliothèques ne possède pas les connaissances nécessaires pour les utiliser d'une manière efficace. De ce fait, les programmes et les connaissances sur leur utilisation doivent être accessibles par des non informaticiens et surtout par les spécialistes du domaine pour lequel ces programmes sont écrits. La solution est alors de développer des systèmes informatiques pouvant prendre en charge la gestion de l'utilisation de ces bibliothèques, affranchissant un utilisateur de ce savoir-faire et lui permettant de se focaliser sur l'interprétation des résultats. Ces systèmes sont dits systèmes de pilotage et permettent la réutilisation intelligente de programmes. Ces systèmes répondent bien au besoin de mise en commun de services, ressenti dans le cadre de plusieurs domaines à savoir l'imagerie médicale. A titre d'exemple, nous citons le suivi de chimiothérapie sur ostéosarcome qui utilise une méthode d'Analyse Factorielle de Séquences d'Images Médicales (AFSIM) [2].

Un système automatique de pilotage permet ainsi à des médecins d'exécuter les programmes qui composent une chaîne de traitement, de vérifier la consistance de méthodes d'analyse d'images, de comparer des algorithmes, d'évaluer des résultats, de revenir sur certains paramètres, et de les réajuster.

Les médecins utilisateurs voulant bénéficier des services du pilotage peuvent être dans des sites distincts, les programmes et les connaissances sur leur utilisation peuvent être écrits par des personnes différentes et donc situés dans des machines distantes. Ainsi, l'étude de la distribution de ces systèmes s'avère intéressante puisqu'elle permet soit la simple consultation distante de bases de connaissances, soit la construction collaborative de bases de connaissances, soit le lancement d'une requête sur un système à base de connaissances distant, avec des données locales.

Notre objectif est de proposer des solutions de coopération et de partage qui permettent à des équipes de mettre en commun des programmes d'imagerie médicale et les connaissances sur leur utilisation et de bénéficier ainsi des travaux d'autres équipes.

Dans cet article, nous commençons par définir les systèmes de pilotage de programmes et l'intérêt de leur distribution. Dans la section 4, nous proposons une architecture distribuée pour un système de pilotage illustrée par un premier scénario pour le traitement des requêtes.

2. Systèmes de pilotage de programmes

Un système de pilotage est un Système à Base de Connaissances (SBC) qui assure la sélection et l'enchaînement de programmes dans différentes configurations, grâce au raisonnement de son moteur et aux connaissances contenues dans sa base. Ceci permet la réutilisation «intelligente» [6,8] de programmes, sans modifier leur code et rend ces programmes accessibles à des utilisateurs non-spécialistes des techniques et des algorithmes implantés dans les programmes. Des environnements de pilotage ont vu le jour [9] afin d'offrir une automatisation de l'utilisation de diverses bibliothèques de programmes.

Un tel système est composé comme tout SBC d'une base de connaissances qui contient le savoir-faire sur l'utilisation des programmes à piloter et d'un moteur de pilotage qui utilise ce savoir-faire pour construire un plan d'exécution des programmes et l'exécuter pour obtenir les résultats ; le moteur doit avoir toutes les connaissances pour planifier automatiquement les programmes à exécuter, lancer leur exécution, produire des résultats, les évaluer et savoir quelles actions de correction entreprendre (re-planification ou re-exécution de l'étape courante) en cas de mauvaise qualité. De plus, il comporte aussi une bibliothèque de programmes à piloter, adaptés à un domaine d'application précis, un ensemble de données à traiter (par exemple, des images dans le cas de l'imagerie médicale) et une interface graphique permettant à un utilisateur d'exprimer son objectif, de suivre l'exécution et de visualiser les résultats.

3. Pilotage distribué

Les systèmes de pilotage distribués offrent des services, d'une part, à des utilisateurs distants qui souhaitent utiliser les facilités de résolution optimale du pilotage pour traiter leurs données et, d'autre part, à des experts et des concepteurs pour diffuser et partager des programmes. Ces derniers doivent travailler de façon collaborative, c'est-à-dire pouvoir consulter des informations sur les programmes existants, créer de nouveaux SBC à plusieurs ou compléter un SBC existant en y introduisant de nouveaux programmes et de nouvelles connaissances.

Les environnements de pilotage ont originellement été conçus pour être mono-site. Cependant, de plus en plus, les constituants d'un environnement de pilotage (cf. § 2) peuvent être localisés sur différents sites. Non seulement les codes et/ou les données ne sont pas obligatoirement sur le même site, mais des parties du moteur lui-même peuvent être délocalisées, pour des raisons de performances, de particularités matérielles, etc. [5]. Un environnement mono-site ne suffit plus, car il implique l'installation de tous les composants en particulier, les programmes et la base de connaissances complète sur ce site. Or, ceci n'est pas toujours possible ni souhaitable. D'une part, l'installation et la maintenance de codes reste un lourd problème qui nécessite temps et compétence. S'ajoute à cela le fait que les programmes s'avèrent souvent peu portables et sont difficiles à installer (car nécessitant l'appel à d'autres utilitaires ou dépendants d'environnements de développements, ou de compilateurs). D'autre part, lorsque des utilisateurs sur différents sites créent des (parties de) bases de connaissances, ils développent de ce fait une compétence propre qu'il est intéressant de mettre facilement à disposition d'une communauté utilisatrice de techniques similaires. Le rapatriement de ces connaissances sur tous les sites utilisateurs poserait des problèmes de cohérence et de maintenance.

La mise en place d'architectures et de services pour la résolution distribuée de problèmes de pilotage peut s'effectuer selon deux axes. D'une part, la mise en place de serveurs de connaissances permet le partage et la diffusion des connaissances sur de multiples sites clients. D'autre part, la mise à disposition d'un même environnement de pilotage à un ensemble d'utilisateurs permet de mettre en commun les connaissances développées sur différents sites. Ainsi, chaque site permet l'utilisation de façon transparente des ressources (programmes et connaissances) d'autres sites. Chaque site devient client des compétences d'autrui et serveur de ses propres compétences. Par conséquent, si des programmes liés à une même problématique sont développés dans des équipes disséminées, le pilotage distribué peut permettre une résolution coopérative de problèmes, dans laquelle chaque étape représente le savoir-faire d'une équipe. Il permet aussi à des chercheurs de confronter leurs expériences et d'enrichir leurs résultats.

Pour cela, nous avons étudié différentes modalités de distribution d'un système de pilotage. Différents cas

de distribution sont donc à envisager : par exemple, un utilisateur peut lancer une requête de traitement sur des données et des programmes locaux, avec une base de connaissances et un moteur distants. Chaque forme de distribution engendre des problèmes différents dus à la complexité des données à transférer, à l'hétérogénéité des langages et des environnements de développement, à des besoins spécifiques de ressources (matérielles ou logicielles) pour l'exécution de certains programmes, à la gestion de la cohérence des connaissances d'usage sur les mêmes programmes en différents sites, etc. Par exemple, un moteur sur un site et une base de connaissances sur un autre site engendrent des accès coûteux [5].

4. Architecture d'un système de pilotage distribué

4.1. Choix d'une technique de distribution

Afin d'implanter une architecture adéquate pour notre système de pilotage distribué, nous avons étudié des techniques réseaux à savoir la technologie Peer-to-Peer [4] et le Grid Computing [3,4] et déduit ce qui suit : les grilles se sont intéressées aux services et applications relativement sophistiqués, reliant peu de sites dans des collaborations engagées dans des applications scientifiques complexes. En revanche, les systèmes P2P font intervenir beaucoup plus de participants, et offrent des services non sophistiqués, limités et spécialisés, tels que le partage de fichiers.

Dans notre cas, le système de pilotage distribué doit être puissant, ne doit pas se limiter à de simples opérations d'échange de fichiers mais plutôt doit offrir des fonctionnalités complexes telles que l'exécution de programmes distants et l'accès en différents modes aux données et aux fichiers de connaissances, et doit pouvoir paralléliser des traitements afin de les accélérer, comme par exemple dans le cas où nous voulons appliquer le même algorithme sur les différents segments d'une image radio d'un os (cas de l'ostéoporose) sachant qu'elle peut comprendre entre 2000 et 4000 segments.
Par conséquent, nous avons opté pour une architecture GRID puisqu'elle offre des stratégies plus riches permettant d'accélérer les traitements et d'augmenter la collaboration.

4.2. Architecture proposée

L'architecture proposée est alors une « petite » grille équipée d'un entrepôt de méta-données servant à la localisation des différentes ressources et gérée par un système multi-agents à base d'agents mobiles qui ont pour rôle de migrer d'un site à un autre pour exécuter des programmes distants. Cette architecture, donnée dans la figure 1, comprend un serveur de pilotage (cf. § 4.3) qui joue le rôle d'un intermédiaire entre l'utilisateur final et le système de pilotage composé de serveurs de programmes, de serveurs de connaissances et d'un moteur de pilotage.

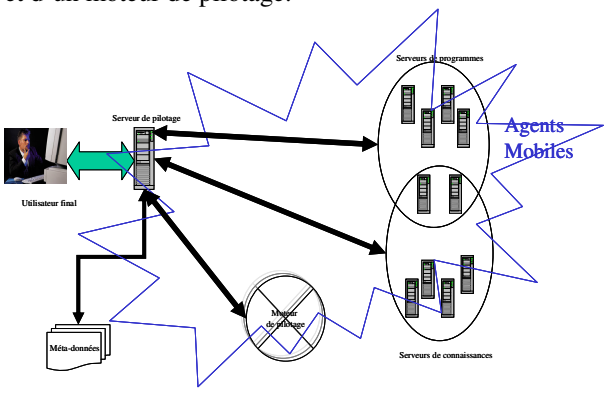

Figure 1. Architecture d'un système de pilotage distribué.

4.3. SPI : un serveur de pilotage via le Web

Au cours de travaux antérieurs nous avons d'une part, développé un prototype de serveur de pilotage, nommé SPI [1], et d'autre part, construit deux bases de connaissances de test sur l'indexation et la reconstruction 3D [7] et l'imagerie médicale (en MatLab) pour la détection de l'ostéoporose. Il est à noter que les programmes de détection de l'ostéoporose sont écrits par Sylvie Sevestre-Ghalila et sont basés sur une approche de morphologie mathématique et sachant que l'étude des données inhérentes au diagnostic de la maladie de l'ostéoporose a été faite en collaboration avec le service de Radiologie de l'hôpital Charles Nicolle de Tunis.

SPI permet la consultation et la modification à distance de telles bases et l'accès authentifié de différents utilisateurs. Il s'occupe du traitement de leurs requêtes, rapatrie les données, délègue le traitement, récupère les résultats et les renvoie à l'utilisateur. SPI est en cours de développement pour améliorer la gestion des accès concurrents et de la cohérence des bases de connaissances.

Ce serveur joue le rôle d'un intermédiaire entre le système de pilotage et l'utilisateur. Ainsi, il communique avec les autres composants et met en place un système d'agents mobiles pour les traitements complexes de résolution de requêtes ou de mise à jour des différents constituants du système. Pour la

migration des agents, la politique actuelle est de laisser les programmes appelés sur leurs localités respectives, de faire migrer les données sur le site où se trouvent les programmes et de récupérer les résultats pour les transmettre au demandeur. Mais le serveur devra pouvoir dans le futur mettre en œuvre différents scénarii de distribution (migration des programmes et des données, par exemple).

4.4. Modèle d'Agents Mobiles

Notre système a besoin au moins de deux types d'agents mobiles, agent « Solveur » et agent « Evaluateur » qui se caractérisent par un comportement et un ensemble de connaissances.

Comportement des agents. L'agent « Solveur » est créé au niveau du serveur de pilotage, il fait le parcours des programmes planifiés par le moteur en lançant leur exécution et en stockant dans son contexte le résultat de chaque itération et les paramètres d'exécution pour les opérateurs suivants. Quant à l'agent « Evaluateur », il est créé au niveau des sites de programmes, s'intéresse à la phase d'évaluation du résultat d'exécution d'un opérateur ou d'un ensemble d'opérateurs et stocke dans son contexte le résultat du jugement. Le nombre d'agents « Evaluateurs » dépend du nombre d'évaluations nécessaires.

Connaissances des agents. L'agent « Solveur » a comme connaissances :

- Une liste de programmes distants à exécuter ainsi que leurs arguments respectifs.
- La liste correspondante des sites de programmes à visiter.
- La liste de ses accointances, à savoir les agents « Evaluateurs » avec lesquels il communique.

Quant à l'agent « Evaluateur », il a comme connaissances :

- Le résultat à évaluer.
- Le site vers lequel il doit migrer (site de l'utilisateur si l'évaluation est manuelle ou site du moteur si l'évaluation est automatique).
- La liste de ses accointances à savoir le ou les agents « Solveurs » avec qui il communique.

5. Scénario pour la résolution des requêtes de médecins

Le scénario suivant décrit la façon de résoudre une requête en utilisant le moteur de pilotage PEGASE de l'INRIA. Ce moteur se base sur différents critères implémentés par des règles de production qui jouent un rôle important au cours du raisonnement, i.e. pour choisir entre différentes alternatives (critères de choix), pour adapter l'exécution des programmes (critères d'initialisation), pour diagnostiquer la qualité des résultats (critères d'évaluation) et pour réparer une mauvaise exécution (critères de réparation et critères d'ajustement). Ces règles sont écrites par des experts sur l'utilisation des programmes.

Une fois connecté au serveur de pilotage SPI, le médecin peut lancer sa requête tout en spécifiant la base de connaissances à interroger en la sélectionnant à partir d'une liste ainsi que les données en entrée. Recevant la requête, le serveur la transforme en un objectif de pilotage de programmes c'est-à-dire en une fonctionnalité compréhensible par le moteur PEGASE, ensuite, vérifie la cohérence des données en entrée avec celles de la fonctionnalité du point de vue nombre et types et enfin, la communique au moteur.

Considérons par exemple, une requête de détection de l'ostéoporose par une approche de morphologie mathématique. En se référant au contenu des méta-données, le serveur déduit qu'elle peut être résolue grâce à la fonctionnalité « Morphology » réalisée par l'opérateur composite (combinaison de programmes) « OsteoMorpho » et il commence par la localisation des fichiers de connaissances à utiliser pour cette requête, par exemples, « osteoporose.kb » le fichier décrivant cette base, « primitives.yakl » et « composites.yakl » qui décrivent respectivement les opérateurs primitifs (élémentaires correspondant à des programmes) et les opérateurs composites. La figure 2 représente un exemple de méta-donnée concernant les programmes de détection de l'ostéoporose par une approche de morphologie mathématique. Cette méta-donnée est décrite à l'aide de RDF et fournit l'auteur, la fonctionnalité, les différents fichiers et leurs sites.

Une fois ces fichiers sont localisés, le moteur de pilotage peut commencer la phase de planification de l'exécution des différents programmes distants c'est à dire construire un plan ou une partie de plan d'exécution de ces programmes.

```
<Rdf : Ostéoporose Morphologie Mathématique>
<Rdf : Author> Sevestre, Sylvie </Rdf:Author>
<Rdf : Functionality> Morphology </Rdf :
Functionality>
<Rdf : Operator> OsteoMorpho </Rdf : Operator>
<Rdf : KB> osteoporose.kb Rdf :
siteKB="http://hellix.inria.fr" </Rdf:KB>
<Rdf : Yakl> composites.yakl Rdf :
siteYakl="http://www.ensi.tn"</Rdf:Yakl>
<Rdf : Yakl> primitives.yakl Rdf :
siteYakl="http://ara.inria.fr"</Rdf:Yakl>
</Rdf : Ostéoporose Morphologie Mathématique >
```

Figure 2. Méta-donnée sur les programmes de détection de l'ostéoporose par une approche de morphologie mathématique.

Pour ce faire, le moteur de pilotage PEGASE commence par la décomposition de l'opérateur « OsteoMorpho » en sous opérateurs de plus en plus concrets (composites ou primitifs).

La figure 3 représente un graphe de la base de détection de l'ostéoporose montrant le plan d'exécution des programmes, généré par le moteur PEGASE pour répondre à cette requête.

« OstéoMorpho » se décompose en une séquence d'opérateurs : *lecture – squelettisation – analyse*. Pour pouvoir décider du nombre d'agents solveurs et du moment de passage à l'exécution des opérateurs planifiés, nous allons tenir compte lors de la planification de :

- la dépendance entre les programmes c'est à dire si les (une partie des) sorties de l'un sont des entrées pour l'autre ;
- l'évaluation c'est-à-dire que la planification s'arrête uniquement quand elle rencontre un opérateur nécessitant l'évaluation de son résultat.

Seul l'opérateur lecture est primitif, mais il ne nécessite pas d'évaluation ainsi PEGASE poursuit son chemin en décomposant « squelettisation » en deux opérateurs alternatifs : *SqueletteBin* | *SqueletteGris*. Le choix de l'un ou de l'autre des opérateurs peut être effectué par l'utilisateur ou par PEGASE lui-même, selon les règles de choix décrites.

- Si le choix est automatique, selon des paramètres et des critères de choix PEGASE décide de l'opérateur à planifier. D'ailleurs, on peut envisager un opérateur par défaut.

- Si le choix est effectué par l'utilisateur, le moteur envoie au serveur la demande de choix qui peut être sous la forme d'une question « Voulez vous effectuer une squelettisation binaire ou une squelettisation en niveau de gris ? ». La réponse de l'utilisateur sera transmise au moteur pour qu'il prenne en considération ce choix dans sa planification. Supposons que selon le choix, le programme *SqueletteBin* a été planifié.

A ce niveau, la squelettisation nécessite une évaluation, donc PEGASE informe le serveur de pilotage des noms des programmes à exécuter (*lecture* et *SqueletteBin*), de leurs emplacements, ainsi que du type de l'évaluation des résultats (pour SqueletteBin). Par la suite, un agent « Solveur » sera créé et informé de tous ces paramètres (programmes, emplacements et type de l'évaluation). Cet agent commence par la migration vers le site distant du programme « *lecture* » ayant dans son contexte, outre les paramètres précédents, les données à traiter.

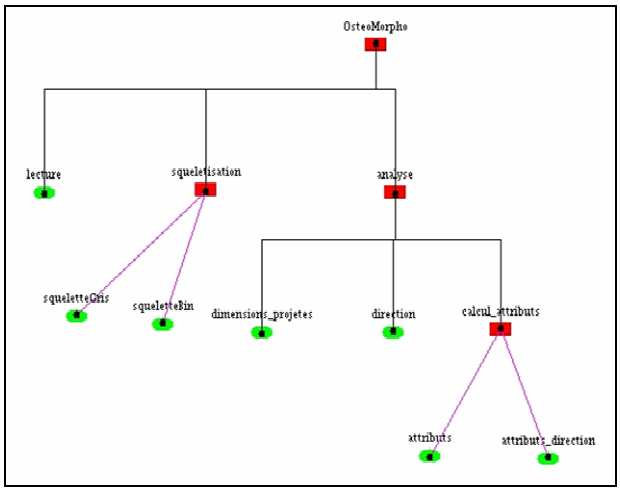

Figure 3. La base de détection de l'ostéoporose.

Une fois ce programme est exécuté, l'agent « Solveur » migre vers le site du programme suivant (*SqueletteBin*) portant avec lui le résultat de « *lecture* » et les paramètres nécessaires à l'exécution du nouveau programme. Une fois l'exécution du plan ou de la partie du plan fournie par Pégase est terminée, il est temps de passer à la phase d'évaluation des résultats prise en charge par l'agent « Evaluateur ». Cet agent sera informé par le « Solveur » du résultat et du type de l'évaluation pour lui permettre de décider du lieu de migration :

- Si l'évaluation est automatique, il migre vers le site du moteur de pilotage ;
- Si l'évaluation est manuelle, il migre vers le serveur de pilotage pour demander l'avis de l'utilisateur. L'« Evaluateur » termine par envoyer ce jugement au moteur.

* Si le jugement est positif, le moteur continue sa planification avec l'opérateur composite « *Analyse* » et transmet son plan à l'agent « Solveur ».

* Si le jugement est négatif, le moteur va utiliser des critères de réparation ou d'ajustement des paramètres pour décider de relancer la planification ou de relancer l'exécution du même plan.

- S'il traite des critères de réparation, le moteur va réparer l'ancien plan, par exemple, en ajoutant un opérateur optionnel non considéré précédemment ou en optant pour un deuxième opérateur s'il s'agit d'un choix. Dans ce cas, nous avons une re-planification et le « Solveur » sera informé du nouveau plan et va se comporter de la même manière qu'avec l'ancien plan.

- S'il traite des critères d'ajustement et selon ces critères, le moteur réajuste quelques paramètres de programmes. Ce réajustement peut être automatique (fait par le moteur) ou manuel. Dans le cas du

réajustement manuel, le moteur transmet une demande d'ajustement de paramètre au serveur pour être traitée par l'utilisateur, comme par exemple : « Pour réparer le plan, voulez vous réajuster le paramètre P de 10 à 9 ou à 8 ? ». Le serveur remet le paramètre ajusté directement au « Solveur » qui re-exécute son programme en tenant compte de la mise à jour apportée sur le(s) paramètre(s).

Ceci se répète jusqu'à arriver au dernier programme (*attributs* ou *attribus_direction*). L'agent « Solveur » exécute cet opérateur tout en étant informé qu'il représente le dernier programme afin qu'il puisse revenir au serveur de pilotage muni du résultat final du pilotage (la réponse à la requête). A son tour le moteur expédie le plan établi au serveur. Finalement, ce dernier transmet le tout à l'utilisateur final.

6. Conclusion

Les environnements de pilotage ont originellement été conçus pour être mono-site. Cependant, leur intérêt et la nature de leurs composants ont rendu nécessaire leur distribution afin d'offrir des services à des utilisateurs distants, médecins dans notre cas, qui souhaitent utiliser les facilités de résolution optimale du pilotage pour traiter leurs données ou de permettre des accès à des experts et à des concepteurs pour diffuser et partager des bibliothèques de programmes. Dans cet article, nous avons proposé une architecture d'un système de pilotage distribué combinant une technologie réseau (Grid Computing) pour la distribution et une technologie IA (Agents mobiles) pour le traitement distribué. Puis, nous avons montré comment un tel système se comporte pour résoudre une requête de détection de l'ostéoporose.

L'intérêt de ce travail a été ressenti par des équipes diverses dont l'équipe du CHR d'Orléans qui s'intéresse aussi à la détection de l'ostéoporose. Un travail prochain portera sur l'intégration des programmes de cette équipe dans notre système distribué ainsi qu'à l'étude de la possibilité de sa mise en œuvre avec des ontologies pour simplifier la recherche des connaissances distribuées dans la grille du pilotage

Remerciements

Nous remercions Sylvie Sevestre-Ghalila, Maître de conférences à l'Ecole Supérieure de Statistique et Analyse de l'Information de Tunis et à l'Université de Paris 5, l'auteur des programmes de détection de l'ostéoporose par une approche de morphologie mathématique pour ses aides judicieuses et pour l'intérêt qu'elle porte sur notre travail.

Bibliographie

[1] Ben Salah T., *"Conception et réalisation d'un outil sécurisé pour la gestion des ressources distribuées sur Internet : application au pilotage de programmes"*, Rapport de stage de 3ème année, ENSI Tunis, juin 2000.

[2] Crubézy M., Aubry F., Moisan S., Chameroy V., Thonnat M. et Di Paola, R., *"Managing complex processing of medical image sequences by program supervision techniques"*, Proceedings of SPIE Medical Imaging 1997, vol. 3035-85, pp. 614-625, Newport Beach, CA, February 1997.

[3] Foster I. et al., *"The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications and High Performance Computing*, 15(3), 2001.

[4] Foster I. et Iamnitchi A., *"On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing"*, *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03),* February 2003, Berkeley, CA.

[5] Lejouad W., *"Etude et application des techniques de distribution pour un générateur de systèmes à base de connaissances"*. Thèse, université de Nice, Nice, juillet 1994.

[6] Moisan S., Lejouad-Chaari W., *" Réutilisation intelligente de programmes de vision en environnement distribué "*, *TAIMA 2001, 2èmes ateliers Traitement et Analyse d'Images : Méthodes et Applications*, Hammamet, Tunisie, Octobre 2001.

[7] Ouali A., *"Vers la conception d'une base de connaissances pour le pilotage de programmes d'analyse d'images"*, Rapport de stage de 3ème année, ENSI Tunis, juin 2000.

[8] Thonnat M., Moisan S., *"What can Program Supervision Do for Software Reuse ?"*, IEE Proceedings-Software. *Special Issue on Knowledge Modelling for Software Components Reuse*, 147(5) : 179-185, October 2000.

[9] Thonnat M., Moisan S., *"Knowledge-Based Systems for Program Supervision"*, Proceedings of KBUP'95, pp 3-8, INRIA Sophia Antipolis, France, Novembre 1995.