Agents Logiciel : Quel est le coût de la distribution ?

Software Agents and distribution : How much are you ready to pay ?

Madeleine GIRARD-FAUGÈRE
Université PARIS VI
LIP6-POLE_IA,
4 place Jussieu, Boîte 169
75252 Paris Cedex 05
girard@laforia.ibp.fr

Résumé

En tirant les leçons de l'implantation d'un logiciel ADAM, *qui est décrit dans cet article, nous répondons à la question du titre par : CHER ! Partant de scénarios, puis en nous plaçant dans le cadre plus général de pilotage de programmes en environnement hétérogène réparti, nous confrontons les coûts d'une approche centralisée par rapport à une approche distribuée. Suite à une description des moyens techniques, nous terminons par une discussion mettant à jour les problèmes de granularité dans le concept d'Agent (Logiciel).*

Mots Clef

Intelligence Artificielle, pilotage de programmes, composants logiciels, agents, systèmes distribués.

Abstract

Based on the lessons on a software implementation called ADAM, *that is presented in this paper, we address the question of the title by : EXPENSIVE ! Starting from two scenarios, we first try to characterize the costs of a distributed approach versus a centralized approach for the program supervision in a distributed environment. Then we describe* ADAM*'s technical realization, and conclude on a discussion pointing out the granularity problems in the (Software) Agent concept.*

Keywords

Artificial Intelligence, program supervision, software components, agents, distributed systems.

1 Introduction

La présentation de ce papier débute par deux scénarios caractérisant les difficultés rencontrées par un utilisateur lors de l'exploitation d'un environnement hétérogène réparti.

1.1 Deux scénarios

Nous vous proposons de réaliser un ensemble d'opérations dans un environnement constitué d'une seule machine (scénario 1) puis de plusieurs (scénario 2). Tout en minimisant leur temps d'exécution, nous vous demandons d'effectuer un calcul nécessitant le logiciel «Maple" ainsi qu'un programme de calcul, d'insérer le résultat obtenu dans un fichier LaTeX, de le compiler puis de l'imprimer.

« 1.1.1 Différents Environnements

Chaque scénario propose une architecture matérielle différente :

Scénario 1 :

Vous avez à votre disposition une station Sun avec 128 Mb de RAM reliée à une imprimante nommée IP7 ainsi que les logiciels suivants : le logiciel «Maple», le programme de calcul et le package LaTeX.

Scénario 2 :

Vous avez à votre disposition plusieurs Sun en réseau avec 32 Mb de RAM chacun. Le premier (MS2) possède le logiciel "Maple» d'installé, le second (MS5) le logiciel «Maple» ainsi que le programme de calcul, puis le troisième (MS20) le package LaTeX. Seul MS2 est configuré afin de permettre l'impression sur la seule imprimante disponible, IP7.

1.1.2 Les réponses

Scénario 1 :

1. Activation du logiciel «Maple» sur vos données.
2. Conversion du résultat obtenu au format reconnu par le programme de calcul grâce à l'application «XYX» écrite par votre collègue.
3. Lancement du programme de calcul.
4. Insertion des résultats dans un fichier «TeX» grâce à la commande Ctrl x-i d'«émacs».
5. LaTeX, génération du PostScript par la commande «dvips -o» et impression par la commande «lpr -PIP7».
6. Récupération de l'impression.

Scénario 2 :

1. Activation du logiciel «Maple» sur MS5 afin de pouvoir enchaîner sur le programme de calcul : «échec - Espace mémoire insuffisant!» Une exécution gourmande en mémoire est en cours d'exécution et vous ne voulez l'interrompre.
2. Transfert (par ftp ou rcp) de vos données sur MS2, puis activation du logiciel «Maple».
3. Recherche de l'application «XYX» écrite par votre collègue : elle est présente sur MS20 ce qui nécessite encore une migration des résultats.
4. Conversion effectuée par le programme «XYX» puis retour des résultats sur MS5.
5. Lancement du programme de calcul.
6. Insertion des résultats dans un fichier «TEX».
7. Migration des résultats sur MS20.
8. LATEX puis génération du PostScript.
9. Migration du fichier PostScript sur MS2 et lancement de l'impression.
10. Enfin vous récupérez votre impression !

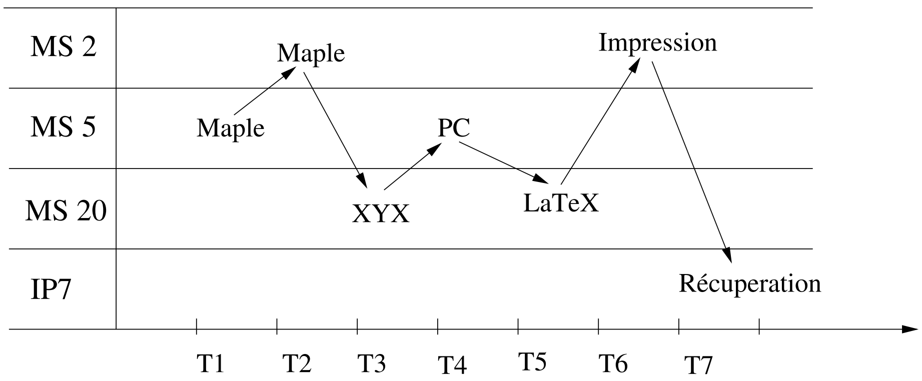

FIG. 1 - *Les différentes opérations du scénario 2*

1.2 Conclusion

Ces scénarios posent, pour ce qui nous concerne ici, la problématique de la composition de composants logiciels répartis par agencement et placement de ces composants dans un environnement constitué d'une ou de plusieurs machine(s) en réseau. Ils soulèvent plus particulièrement les problèmes de distribution, de coopération et de contrôle d'entités distribuées dans un but de création d'applications réparties.

Les systèmes multi-agents [Fer95] constitués d'entités réparties en interactions sont des outils adéquats pour la résolution de manière conviviale des problèmes soulevés par ces scénarios.

L'utilisation de cette technique, nous amène à définir ce qui pourrait être un agent dans le cadre de notre problématique. Nous en arrivons au fait que le concept d'«agent» peut recouvrir plusieurs entités totalement différentes, comme par exemple :

– chaque ou toutes les machine(s),

- le réseau,
- la mémoire,
- chaque ou tous les composant(s) logiciel(s),
- l'imprimante,

Nous repoussons la détermination de ce que l'on appellera «agent» à la section 4, les sections 2 et 3 servant à définir les moyens de cette détermination.

1.3 Problématique

De nos jours les logiciels ne sont plus monolithiques à fonctionnalités figées, mais résultent d'une composition modulaire variable répondant aux exigences de l'utilisateur pour la résolution d'un problème spécifique dans le domaine, par exemple, du calcul scientifique [DGS94] [DJR95] ou du traitement d'images [CM96] [DJ94] [LOG97]. Cette composition modulaire naît de l'assemblage et des interactions d'un ensemble de composants spécialisés distribués sur le réseau. Bien que favorisant la réutilisation de composants spécifiques existants et proposant une meilleure exploitation des ressources pour de meilleures performances à l'exécution, la composition d'applications réparties pose les trois problèmes suivants :

1. *la configuration*: le choix et l'agencement des composants logiciels nécessaires à la résolution d'un problème de l'utilisateur. La configuration créée ce que nous appelons une composition fonctionnelle.
2. *la résolution* : le placement des composants logiciels en tenant compte de la dynamicité de l'environnement notamment de la disponibilité des ressources, conduisant à une composition structurelle.
3. *le contrôle* de la configuration, de la résolution, de la mise en œuvre de l'application répartie, de l'évaluation des ressources, de l'exécution d'une application répartie, de son adaptation en environnement dynamique et enfin de sa réutilisation.

Des recherches concernant la problématique du pilotage de programmes sont actuellement en cours. Dans le domaine des systèmes à base de connaissances, [VMT96] et [vdEvHT95] traitent plus particulièrement les problèmes liés à la configuration ainsi qu'à son contrôle par le biais d'une approche «méta-contrôle» [Pit90]. Nous abordons ici cette problématique de manière différente en résolvant plus particulièrement les problèmes liés à la mise en œuvre de programmes distribués dans un environnement réparti ainsi que les problèmes liés à leur évolution en environnement dynamique, laissant une grande partie du contrôle de la configuration à l'utilisateur.

Les problèmes liés à la composition d'applications réparties en environnement distribué et dynamique sont traités, dans cet article, sous l'angle de la problématique générale du *contrôle* et plus précisément du *pilotage de programmes* au sens propre du terme. Nous utilisons le terme de contrôle en tant que synonyme du terme pilotage de programmes. Le contrôle d'entités distribuées étant indépendant de la distribution des entités elles-mêmes, il peut s'exercer de manière plus ou moins distribué voire centralisé. Nous confrontons tout d'abord les avantages et les inconvénients des différentes approches de pilotages de programmes possibles pour la résolution des problèmes de configuration, de résolution et de contrôle de chaînes de composants logiciels avant de discuter une approche particulière en terme de réalisation en nous basant sur le système ADAM [Gir95] [Gir96].

Cet article suit l'organisation suivante : la section 2 propose une confrontation des avantages et des inconvénients de la distribution et de la centralisation des différents problèmes liés aux pilotages de programmes. La section 3 confronte ces approches en terme de réalisation et la section 4 clôt cet article en proposant une réflexion critique des choix effectués pour ADAM ainsi qu'une discussion sur la question ouverte de la détermination d'un agent dans le cadre de ce travail.

2 Distribution vs Centralisation

L'élaboration d'un système permettant la composition de logiciels à partir de composants hétérogènes répartis nécessite une réflexion quant à la prise de position (centralisée ou distribuée) nécessaire à la résolution des différents problèmes rencontrés : choix de la représentation des connaissances, principes de leur mise à jour, automatisation des opérations, construction d'une solution, exécution d'une solution, sauvegarde d'une solution, confrontation d'une solution en environnement dynamique. Pour chacun de ces différents points nous allons confronter les avantages et les inconvénients d'un choix centralisé ou distribué.

2.1 Quelle représentation choisir?

2.1.1 Centralisation des connaissances

Une représentation des connaissances centralisée correspond à une omniscience de l'utilisation des composants logiciels et des ressources. Une connaissance totale et exacte concernant les composants logiciels (existence, localisation, besoin en terme de mémoire et CPU, version (option de compilation)...) et les ressources (mémoire disponible, charge des machines, état du réseau) apporte à l'utilisateur des avantages de facilité et de rapidité d'utilisation pour la spécification d'une chaîne de composants logiciels performante :

- la connaissance de l'ensemble des composants logiciels permet de créer facilement un enchaînement cohérent, résolvant les problèmes d'interopérabilité.
- la connaissance des disponibilités des ressources permet leur exploitation en vue d'une composition performante.

Ainsi, dans le cadre du scénario 2, l'utilisateur avertit par le système du peu d'espace mémoire disponible sur la machine MS5 et des besoins minimaux du logiciel Maple, aurait directement sélectionné la machine MS2 pour le lancement du logiciel «Maple».

La représentation centralisée des connaissances (concernant les composants logiciels et les ressources) permet leur exploitation naturelle par l'intermédiaire d'une interface graphique comme le propose [LOG97]. Les spécifications des compositions sont aisément réalisées par liaisons des composants logiciels impliqués. Ces spécifications peuvent facilement être sauvegardées et être utilisées comme élément de contrôle de futures requêtes : toute spécification incorrecte ou incomplète, c'est-à-dire dont la création de la composition (fonctionnelle et structurelle) correspondante a échoué, évitera l'activation de la phase de résolution pour toute requête identique à celle n'ayant pas donné satisfaction.

Coût

Une représentation centralisée nécessite la définition d'un formalisme pour la représentation des connaissances constituées des composants logiciels et des ressources. Ce formalisme doit non seulement permettre la représentation des caractéristiques de base des composants logiciels comme leur types d'entrée sortie par exemple, mais il doit aussi permettre la représentation d'informations liées à leur mode d'utilisation comme leurs limites théoriques (les problèmes qu'il ne résout pas), le nombre d'utilisateurs maximal autorisés, etc.

Ce formalisme pose le problème de l'abstraction des connaissances que l'on souhaite le plus souvent minimales. Ce degré de minimisation est lié au degré d'«autonomie» du système pour la résolution des problèmes de configuration, de résolution et de contrôle. Certains systèmes à base de connaissances [VMT96] [DJR95] s'attaquant au problème de la méta-modélisation [Pit90] des connaissances et du raisonnement nécessaires à la décomposition d'un problème en sous-problèmes, soulèvent le problème de la spécification des composants en vue de leur réutilisation [Kru92] [MM94] et [vdEvHT95], ce qui les amènent à traiter une très grande quantité d'information.

Intuitivement, pour être efficace cette représentation totale et centralisée doit suivre en temps réel l'existence des composants et la disponibilité des ressources du réseau. Or une telle mise à jour engendre un coût de communication élevé.

Conclusion

Une représentation centralisée est plus facile et plus rapide d'utilisation et de mise en œuvre mais aussi plus coûteuse en terme de mise à jour.

2.1.2 Distribution des connaissances

La spécification d'une représentation distribuée est une solution attrayante mais coûteuse.

Une représentation des connaissances distribuée paraît être une représentation plus proche du composant et de la ressource que ne l'est une représentation centralisée réduisant les communications (objet à représenter-représentation) et permettant la détection en temps réel de l'ajout ou du retrait de composants logiciels, l'évaluation de la disponibilité des ressources ainsi que la détection des pannes machines ou du réseau.

Afin d'exploiter la distribution dans toute sa puissance, cette connaissance doit être détectée de manière transparente à l'utilisateur. Un tel mécanisme implique la distribution physique d'entités sur le réseau, entités qualifiées «d'agents observateurs» possédant les connaissances centralisées relatives à :

- l'utilisation des composants logiciels,
- la spécification de comportements propre à l'agent comme par exemple la détection de la présence (resp. absence) d'un composant logiciel,
- l'évaluation des disponibilités des ressources,
- la détection de pannes réseau.

Cette connaissance bien que distribuée provient d'un pôle centralisateur (l'utilisateur ou le concepteur) ; elle peut être mise en place simplement sous forme de listes équivalentes aux pages jaunes d'Unix [Rif93], notamment en ce qui concerne la détection de composants logiciels sur le réseau.

La distribution des connaissances n'étant soumise à aucun formalisme, se pose le problème du degré de la distribution. Cette connaissance peut être totalement distribuée; dans ce cas, chaque composant logiciel ou comportement d'auto-détection peut être représenté par une entité physique indépendante, ou dans le cas contraire, ils peuvent être regroupés au sein de pôles centralisateurs multi-composants et/ou multi-comportements correspondant à une «hiérarchisation» de la distribution.

La spécification d'une composition nécessite l'utilisation d'une interface; elle représente un pôle centralisateur de connaissance. Cette centralisation nécessite la propagation de toute ou d'une partie de la connaissance distribuée et détectée vers ce pôle, propagation plus coûteuse ou du moins égale en terme de communication que la mise à jour dans le cadre d'une représentation centralisée.

La distribution peut être une nécessité, c'est-à-dire la seule alternative possible d'implantation de programmes: les composants logiciels nécessitant un important espace de stockage ou de mémoire à l'exécution ne peuvent être installés sur un même disque ou une même machine.

La distribution, bien que plus difficile à mettre en œuvre comme nous le verrons dans le paragraphe suivant, présente des avantages non négligeables de parallélisme. Plusieurs actions (opérations ou contrôles) peuvent être effectuées en même temps, indépendamment les unes des autres, au sein de pôles centralisateurs distincts.

Coût

La mise en place de pôles centralisateurs sans possibilités de communication ni d'interaction avec d'autres pôles ne présente que fort peu d'intérêt.

Les différents pôles doivent posséder des capacités de communication et de raisonnement afin de pouvoir coopérer, échanger des informations ou encore négocier des propositions. La mise en œuvre de capacités de communication pose le problème de la spécification du protocole de communication (réalisé par le concepteur) ainsi que la spécification des connaissances des accointances (pôles voisins). Les connaissances sur ces accointances peuvent être spécifiées par le concepteur de manière fixe et non évolutive ou bien dynamiquement participant ainsi à l'autonomie du système. Cette dynamicité nécessite des possibilités de lecture et de détection des adresses des accointances, adresses généralement stockées sous forme de dictionnaires centralisant celles des pôles serveurs [Smi80]. Tout nouveau pôle peut ainsi décider de ses accointances et les avertir de sa propre existence.

Les seules capacités de communication ne sont pas suffisantes à la mise en œuvre d'interactions entre accointances, il faut y adjoindre des capacités des raisonnement [DL91] et de négociation [Mue96] sur les données reçues en vue de l'obtention d'un complément d'information ou de spécification d'actions. Ces capacités de raisonnement et de négociation sont centralisées au sein de chaque agent et impliquent l'utilisation d'une même ontologie afin d'attribuer une même signification aux données échangées [MLF96].

Conclusion

Nous constatons qu'une connaissance soit disant distribuée correspond à une connaissance centralisée au sein de pôles physiquement distribués, la coordination de ses pôles en vue d'une résolution partielle ou totale de la composition coûtant très cher en terme de communication.

2.1.3 Discussion

Indépendamment du coût d'implémentation, le choix d'une représentation centralisée ou distribuée est lié au type de l'application envisagée. Nous caractérisons leurs limites dans un cas extrême d'utilisation représenté par l'évolution en système ouvert :

- l'utilisation d'une représentation centralisée pose des problèmes physiques de stockage d'information et de taille mémoire d'exécution que ne posent pas la représentation distribuée.
- l'utilisation d'une représentation distribuée pose le problème de la limitation des connaissances au niveau des pôles centralisateurs (cf. item précédent), le problème de la convergence de la résolution distribuée de problèmes ainsi que le problème de l'utilité de la connaissance représentée : en effet, quelle est l'utilité de la représentation d'une connaissance stockée sur un site distant fort coûteux d'accès ?

Nous en concluons que pour des raisons d'efficacité, toute la connaissance n'a pas besoin d'être centralisée : l'ensemble des connaissances inutiles à la résolution du (resp. des) problème(s) spécifique(s) au pôle centralisateur peut être et doit être placé sur des pôles distants.

2.2 Mise à jour des connaissances

Un système qu'il soit distribué ou centralisé nécessite une mise à jour des connaissances concernant :

- la spécification de nouveaux composants logiciels voire de nouvelles machines. (Problème résolu de manière centralisée par le concepteur ou bien par des comportements autonomes comme nous l'avons expliqué dans la section précédente).
- la disponibilité des machines, disponibilité variant continuellement au cours du temps.

L'évaluation de la disponibilité des ressources pose le problème du choix des critères sur lesquels va s'effectuer cette évaluation. Son résultat est ensuite propagé vers la représentation centrale dans le cas d'un système centralisé ou vers son pôle serveur dans le cas d'une représentation distribuée. L'intervalle de temps entre l'évaluation, la propagation puis l'utilisation de cette représentation étant non négligeable, la disponibilité réelle des machines ne sera plus la même, engendrant un décalage entre l'environnement réel et sa représentation.

2.3 Automatisation idéale

2.3.1 Les objectifs d'une automatisation

La composition de logiciels de manière manuelle constitue une séquence d'opérations souvent longue et difficile. Une automatisation de ces tâches permet un gain de temps précieux comme le montre [ELS94] [ELST93] au travers la génération automatique de commandes Unix.

Le traitement de tâches annexes par le système permet à l'utilisateur de se concentrer sur l'essentiel, c'est-à-dire la spécification de la composition et non plus la mise en œuvre de l'application répartie. Or, cette phase de spécification peut également être automatisée ne sollicitant de la part de l'utilisateur que la spécification des plus importantes opérations. Cet automatisme implique la spécification de méta-connaissances relatives à la classe de problèmes traités. De plus, plus l'automatisme de la composition est évolué, plus la gestion du système devient complexe, chargée d'exceptions, entraînant la mise au point de règles d'utilisation voire de manuels d'utilisation : «automatisme» doit rester lié à «facilité d'utilisation».

2.3.2 Coût de l'automatisation

Les opérations de spécification de la composition ainsi que celles de connexion et d'envoi de données d'un logiciel à un autre nécessitent la mise au point de règles de spécification pour la composition ainsi qu'un protocole de communication.

Règles de spécification

Lors de la création des règles de spécification, le concepteur doit d'une part s'assurer de leur cohérence puis vérifier l'adéquation entre la composition souhaitée et la composition obtenue [MMdP95].

Ces règles doivent définir un formalisme de spécification donnant à l'utilisateur l'impression de ne spécifier que le minimum et l'essentiel de sa requête.

Gestion des connexions physiques

La connexion des logiciels et la gestion de l'envoi des données nécessite un formalisme connu de tous les composants ou gestionnaires de composants, leur spécifiant les connexions à créer ou à ouvrir ainsi que le ou les destinataire(s) des données et des messages.

2.4 Construction d'une application répartie

En se plaçant dans le cadre de l'utilisation des règles de spécification permettant à l'utilisateur de ne spécifier que certaines fonctionnalités de son application réparties (voir section 3.3.1), la construction d'une application répartie nécessite la détermination des composants non spécifiés, leur agencement ainsi que leur placement sur des machines différentes. Réalisée à partir de spécifications (centralisées), cette phase appelée «résolution» peut s'effectuer de manière centralisée ou distribuée. Nous allons confronter les deux approches possibles.

2.4.1 Résolution centralisée

La résolution de contraintes relatives aux spécifications de l'utilisateur se fait généralement par des algorithmes de planification [MH95]. Cette résolution, utilisant des connaissances totales, permet la recherche d'une solution optimale, c'est-à-dire dans le cadre de notre problématique, l'agencement et le placement favorisant l'exécution la plus rapide.

Outre son optimalité, un algorithme centralisé est rapide car s'exécute sur un même processeur.

2.4.2 Résolution distribuée

La résolution distribuée nécessite la coordination d'un ensemble de pôles serveurs impliquant des communications, des négociations, des retours sur des décisions prises précédemment (backtrack en cas d'erreur), finalement l'échange d'une grande quantité de messages fort coûteux en temps.

Proposant une recherche de proche en proche, elle ne garantit pas l'optimalité de la composition. Par contre, elle permet la prise en considération de connaissances en temps réel, notamment en ce qui concerne l'indisponibilité des machines en cas de pannes réseau.

2.5 Exécution d'une application répartie

Indépendemment d'une construction centralisée ou distribuée, l'exécution des données par les composants peut s'effectuer au fur et à mesure de la résolution, impliquant une perte de temps en cas d'erreur, un backtrack, ce qui pose le problème de la sauvegarde des résultats intermédiaires.

Nous envisageons de séparer la résolution de la construction pour des raisons de rapidité mais également pour des raisons de réutilisation d'une composition comme nous le présentons dans la section suivante.

2.6 Sauvegarde d'une application répartie. Réutilisation

Il est idéal de pouvoir sauvegarder une composition de manière à la réutiliser à volonté et rapidement.

Il n'y a pas de différence entre une sauvegarde de composition au sein d'un système centralisé ou distribué : la sauvegarde est effectuée au niveau des entités distribuées gérant les connexions entre composants. La représentation de la nouvelle composition est propagée au niveau de l'interface de l'utilisateur : elle peut être utilisée directement pour le traitement des données de l'utilisateur ou bien être insérée en tant que nouveau composant logiciel au sein d'une nouvelle application répartie.

2.7 Confrontation d'une application répartie en environnement dynamique

Suite à une variation de la disponibilité des machines, une composition optimale sauvegardée à l'instant τ, ne l'est plus à l'instant $\tau + 1$, moment de la prochaine exécution.

La recherche d'une nouvelle composition à chaque exécution n'est pas concevable car fort coûteuse en temps et en ressources CPU. Ainsi, le gain de temps fourni par la centralisation de la résolution d'une composition est perdu. Nous proposons une solution (présentée sections 3.2.2 et 3.3.3) permettant de capter la dynamicité de l'environnement et proposant une solution optimale à chaque exécution grâce à une phase d'apprentissage.

3 ADAM : une solution semi-centralisée

Nous présentons les choix de centralisation et de distribution effectués pour la réalisation d'ADAM, illustrés par la figure 2 (le lecteur se reportera à cette figure pour avoir une vue globale). Puis la réalisation technique terminera cette section.

3.1 ADAM face à la distribution

ADAM est un système proposant une composition et une gestion automatique d'applications réparties de manière invisible à l'utilisateur.

C'est aussi un système distribué à contrôle semi-centralisé (voir figure 2).

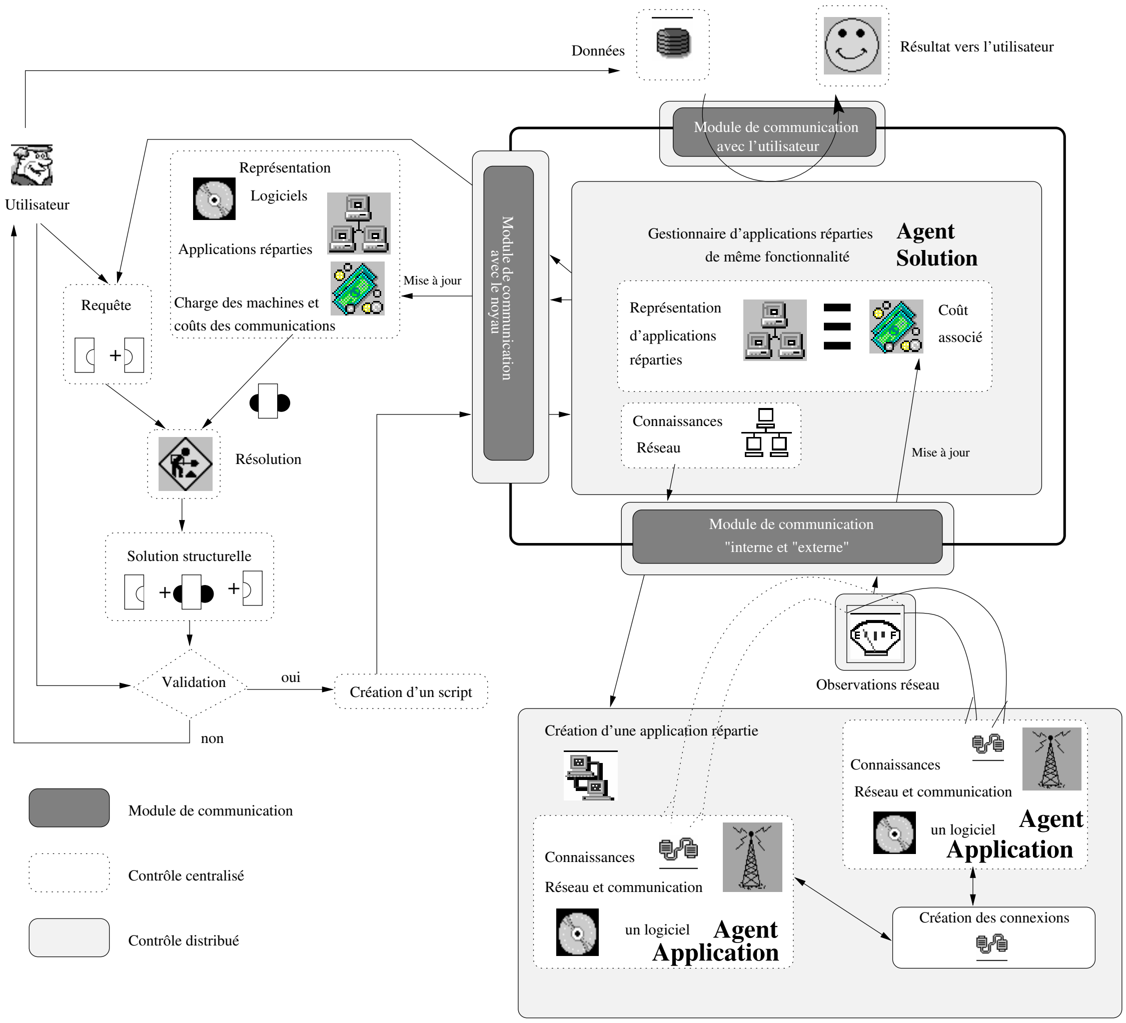

FIG. 2 - *Les différents types de contrôle*

Il est distribué physiquement par un ensemble d'agents en interaction dispersés sur le réseau assurant la gestion des compositions, des données ou bien évaluant les caractéristiques matérielles. Le contrôle est quant à lui semi-centralisé :

– centralisé pour la création d'une composition et la résolution du problème d'allocation de ressources nécessitant des connaissances globales.

– distribué pour ce qui ne traite pas directement de la composition, c'est-à-dire la création des connections physiques, la gestion et l'envoi des données ainsi que l'évaluation des caractéristiques matérielles.

3.2 ADAM contre le scénario 2

Nous allons vous présenter les principes de fonctionnement d'ADAM en spécifiant pour chacune des fonctionnalités la nature du contrôle (centralisé vs distribué). Ces principes de fonctionnement sont exposés en s'appuyant sur l'exemple du scénario 2 présenté en introduction ainsi que sur la figure 2. Le scénario 1 n'est pas traité, car les difficultés qu'il propose sont également présentées par le scénario 2.

3.2.1 ADAM centralise...

...la représentation des connaissances constituée de composants logiciels et du réseau est réalisée de manière centralisée. Elle représente :

– les composants logiciels caractérisés par leur interface (types et configuration d'entrée et de sortie des arguments, éventuellement la principale fonctionnalité du composant comme l'imprimante par exemple).

– les machines par un taux d'occupation fonction de leur disponibilité et de la vitesse du microprocesseur évalué par ADAM.

– les caractéristiques du réseau par les distances intersites également évaluées par ADAM.

...l'automatisation de la composition
Les «τ» indiquent des étapes successives.

τ=0[1] *L'utilisateur décompose* son problème en une chaîne de fonctionnalités (par exemple «Calcul A» - «Calcul B» - «Insertion» - «Impression»).

τ=1 *L'utilisateur instancie* certaines de ces fonctionnalités par des composants logiciels selon les règles de spécification présentées en section 3.3.1. Ces spécifications sont réalisées à l'aide d'une interface et prennent la forme d'un enchaînement de composants logiciels et de fonctionnalités que l'on peut assimiler à des pièces de puzzles. La requête spécifique à l'exemple traité prend la forme suivante, transcrite ici en mode texte pour gain de place : «Maple» - «Calcul» - «Insertion» - «Impression sur IP7».

τ=2 ADAM *instancie* les fonctionnalités restantes, résout les problèmes d'interopérabilité et de placement des composants dans un objectif d'optimisation des performances. Cette instanciation se fait suivant un algorithme de recherche et d'apprentissage (voir section 3.3.2)) utilisant des connaissances relatives aux habitudes de l'utilisateur, aux charges et à la disponibilités des machines ainsi que des connaissances spécifiques à l'algorithme de recherche lui-même. Cette phase de résolution consiste, de manière métaphorique, à la terminaison du puzzle créé par l'utilisateur. En ce qui concerne l'exemple traité, ADAM trouve la composition résultat suivante (les deux fonctionnalités soulignées mettent en évidence des problèmes traités en section 4) : «Maple sur MS2» - «XYX sur MS20» - «Calcul sur MS5» - «Insertion sur MS5» - «LaTeX sur MS20»- «Dvip -o sur MS20» - «Migration des résultats sur MS2» -«Impression sur IP7».

...l'émission de l'ordre de construction sous forme de script vers une entité nommée *Agent Solution* spécialement créée par le noyau central à chaque création d'une nouvelle composition répartie (la figure 3 illustre le contenu d'un Agent Solution).

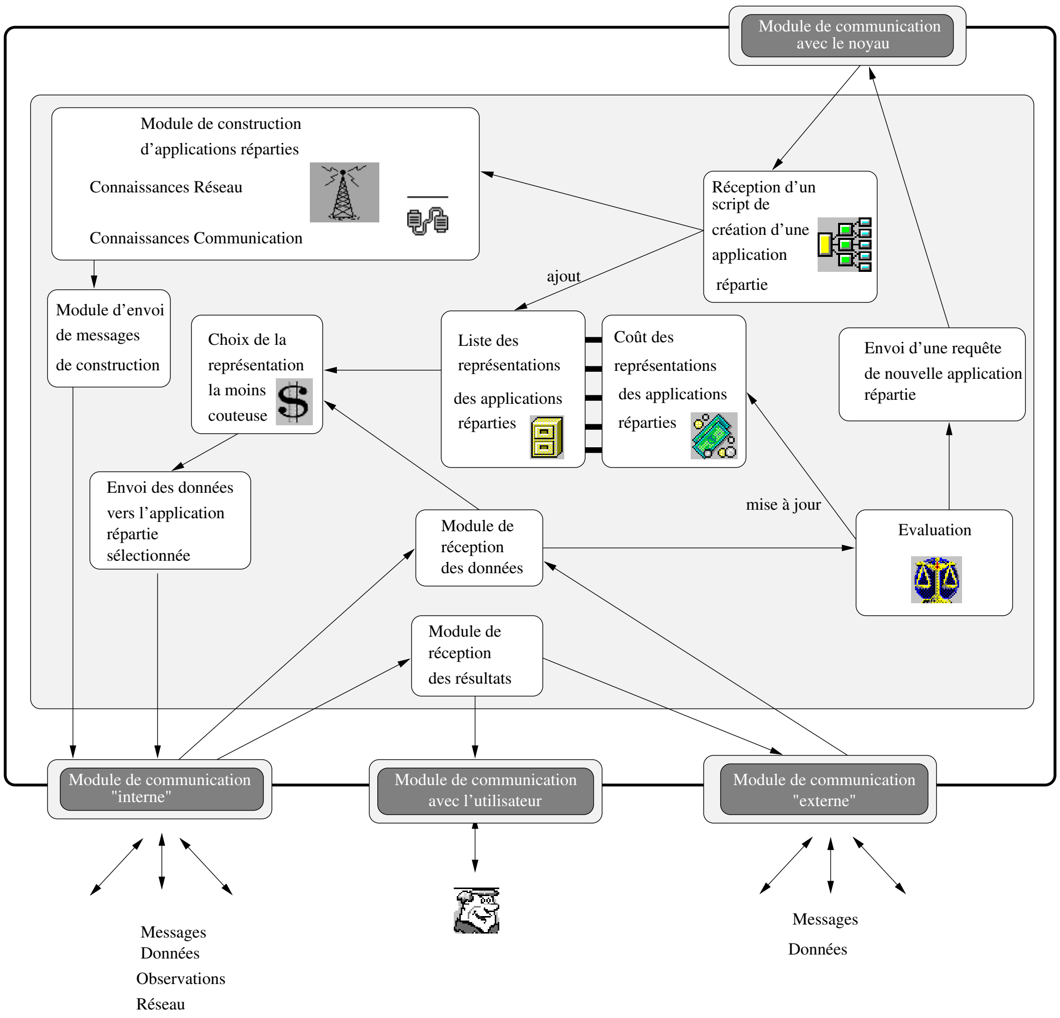

FIG. 3 - *Un Gestionnaire : Agent Solution*

3.2.2 ADAM distribue...

...l'automatisation de la construction physique d'une application répartie.

La création physique d'une composition est sous le contrôle de l'Agent Solution qui gère également les données de l'utilisateur. Cette distribution a l'avantage de permettre la recherche d'une nouvelle composition indépendamment du traitement des données de l'utilisateur.

Après réception du script caractérisant l'application répartie à construire, l'Agent Solution créer les messages d'activation des logiciels correspondant à la solution ainsi que les messages spécifiant les connexions à établir. Ces messages sont ensuite envoyés sur le réseau. La création de l'application répartie est réalisée de proche en proche entre *Agents Application* sur le principe de la continuation.

Chaque composant logiciel est sous le contrôle d'un Agent Application (voir figure 4) jouant entre autres le rôle d'interface entre le réseau et le composant physique, notamment pour l'envoi et la réception de messages vers les (resp. de la part des) Agents Application accointants. La création des connexions se fait de manière distribuée afin de laisser à chaque Agent Application le choix du type de moyens de communication envisagé avec ses accointances : par exemple, dans le monde Unix ces moyens peuvent être des sockets ou des pipes. Cette décision doit être traitée localement puisqu'elle n'utilise que des informations sur la localisation des accointances : on réalise un gain de temps par délégation des prises de décisions.

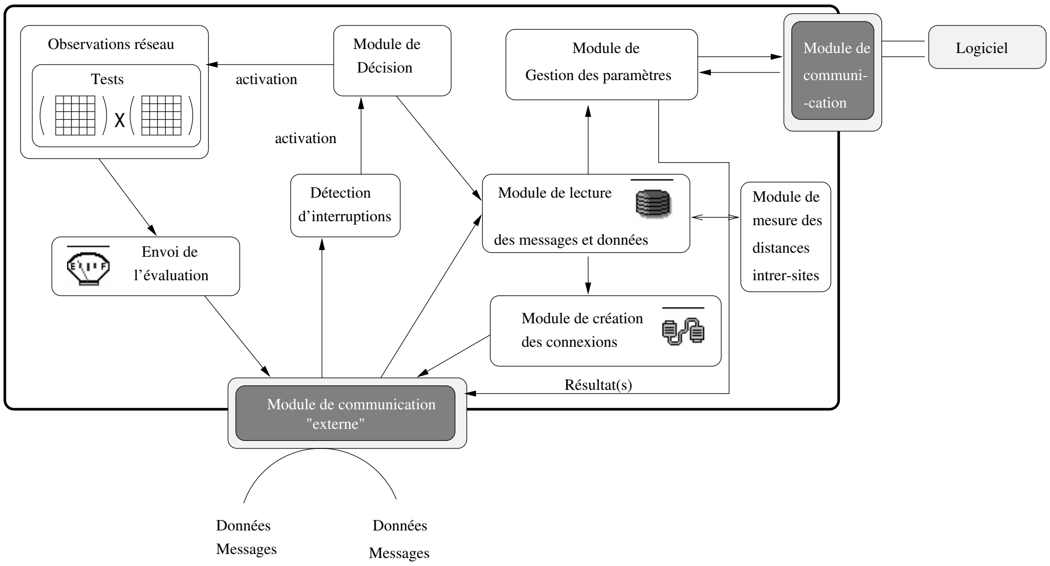

FIG. 4 - *Un Agent Application*

... l'évaluation des disponibilités des ressources et des performances des machines

Afin d'éviter à l'utilisateur de spécifier au système un ensemble de connaissances concernant les caractéristiques techniques et matérielles qui peuvent lui être inconnus, ADAM est muni d'un système d'auto-détection et d'évaluation. Il évalue de manière empirique la vitesse des processeurs ainsi que la charge des machines par le lancement sur les machines cibles, d'un programme de multiplication de matrice 100×100 sur les entiers. Plus ce calcul est coûteux en temps, plus la machine est chargée ou lente : il faut à tout prix éviter son utilisation.

La distribution d'entités physiques (les Agents Application) ainsi que leur contrôle associé (distribué par rapport au noyau mais centralisé au niveau de chaque Agent) permet la définition de comportements locaux limitant l'envoi anarchique de messages concernant :

– le choix du moyen de communication (cf. paragraphe précédent).

– le choix du moment de la mesure des charges des machines.

– la décision de la propagation de cette mesure : seule une modification significative par rapport à la moyenne des dix dernières évaluations est communiquée à (ou aux) Agent(s) Solution, puis propagée au noyau central.

– la mesure des distances intersites en terme de temps de communication. Cette mesure est effectuée à chaque réception de messages. Les Agents Solution avertissent le noyau central seulement en cas de modification majeure, modification pouvant être dûe à un changement de localisation d'une machine sur le réseau par exemple.

– la gestion des versions comme nous le présentons dans la section suivante.

Ces contrôles distribués limitent le nombre de messages, en évitant l'envoi d'ordres et de réponses du (resp. au) noyau central.

... la sauvegarde d'une application répartie par les Agents Solution. Afin de permettre une réutilisation immédiate d'une composition précédemment créée sans solliciter une nouvelle recherche, ces dernières sont sauvegardées. Cette sauvegarde se fait de manière distribuée par maintien des moyens de communication entre les Agents Application impliqués dans la composition.

Les charges du réseau variant au cours du temps, la performance d'une composition n'est pas maintenue. Pour remédier à cet inconvénient, l'Agent Solution peut solliciter une nouvelle composition structurelle en se substituant à l'utilisateur, ce qui l'amène à gérer un ensemble de versions (compositions physiquement différentes) mais assurant les mêmes fonctionnalités. Ils décident comme nous le verrons section 3.3.3 laquelle attribuer à l'utilisateur pour le traitement de ses données.

... le traitement des données de l'utilisateur. Ce dernier est en relation directe avec chaque Agent Solution. Il lui envoie par l'intermédiaire d'une interface les données devant être traitées par la composition fonctionnelle choisie et attend un résultat en retour.

3.3 Réalisation technique

Les solutions proposées pour la réalisation d'ADAM concernant l'évaluation de la disponibilité des ressources, l'automatisme de la construction d'une solution, sa sauvegarde ainsi que son utilisation ayant été précédemment traitées, nous développons dans cette section plus particulièrement la résolution en parallèle et de manière implicite des problèmes d'instanciation et d'ordonnancement des fonctionnalités et des composants logiciels spécifiés par l'utilisateur.

TAB. 1 - *Les caractéristiques et les degrés*

Degré	NOM	FONCTIONNALITÉ
0	Types	Nom
	Configuration	
1	Machine	Machine
2	Site	Site

3.3.1 Création d'une configuration

Après avoir décomposé son problème en sous-problèmes, l'utilisateur spécifie à l'aide d'une interface une chaîne de fonctionnalités déjà partiellement instanciée par des composants logiciels. Cette instanciation partielle correspond aux possibilités de pilotage que possède l'utilisateur sur le contrôle de l'automatisation de la composition par ADAM.

Pilotage graduel de l'utilisateur

Chaque fonctionnalité ou composant logiciel sont caractérisés par un ensemble de caractéristiques (voir tableau 1), auxquelles sont associés des degrés (0, 1, 2) spécifiés par le concepteur. Le degré 0 représente la notion d'obligation signalant que la caractéristique associée doit être spécifiée par l'utilisateur. Par exemple, la fonctionnalité «Imprimante» est caractérisée par un nom (IP7), par une machine (MS2) et par un site (S3), de degrés respectifs (0, 1, 2).

La spécification d'un composant logiciel ou d'une fonctionnalité lors d'une requête nécessite les spécifications de niveau 0. Les caractéristiques de niveau 1 et 2 pouvant être instanciées par le système ou par l'utilisateur selon le contrôle désiré de celui-ci.

L'utilisateur peut à tout moment user de sa suprématie et spécifier les niveaux 0, 1 et 2 pour l'ensemble des composants logiciels voulus, mais dans ce cas ADAM ne garantit pas les performances de l'application distribuée. Si au contraire l'utilisateur restreint son pilotage sur le système en ne spécifiant que les composants et fonctionnalités au niveau 0, il laisse entière liberté au système quant à la sélection et au placement des composants (de préférence les plus disponibles), favorisant ainsi les performances de l'application répartie.

Le pilotage de l'utilisateur sera toujours existant : il est indispensable pour la spécification de la requête puisque nous ne traitons pas le problème de la composition sous une vision système à base de connaissances [VMT96] capable d'analyser et de décomposer un problèmes en sous-problèmes. Nous avons tenté de modéliser le minimum d'informations concernant les composants logiciels pour des raisons de facilité et de rapidité de traitement, mais ce choix implique la capacité de spécification d'une chaîne de fonctionnalités de la part de l'utilisateur.

Règles de spécification

ADAM étant capable de sélectionner certains composants logiciels et d'instancier certaines fonctionnalités, l'utilisateur est alors confronté aux problèmes suivants :

- Quelles fonctionnalités doit-il instancier?

– Quels composants logiciels doit-il spécifier ?

Nous remarquons que l'ensemble des logiciels existants peuvent être regroupés en deux catégories : l'une regroupant les logiciels qui n'affectent pas la sémantique des données traitées, l'autre contenant des logiciels la modifiant. Nous appelons sémantique des données, le sens véhiculé par ces dernières. Par exemple, quelque soit le format de représentation d'une image (gif, jpeg, ras, ...), l'image visualisée reste la même. La signification d'un texte en format TEX, en format Word, en format Texte reste identique quelque soit le traitement de texte utilisé. Par contre, une simple opération de multiplication ou d'addition modifie la signification de la (resp. des) donnée(s) initiale(s).

Nous déclarons que les logiciels modifiant la signification des données doivent impérativement être spécifiés par l'utilisateur, les autres étant sélectionnés par le système. Nous avons dégagé les règles d'utilisation suivantes spécifiant les «degrés de liberté» du système et de l'utilisateur :

Règle 3.1
Si plusieurs composants logiciels sont disponibles pour assurer une même fonctionnalité modifiant la sémantique des données
et
Si l'utilisateur considère ces logiciels comme non équivalents
Alors il doit spécifier un composant logiciel au degré 0.

Règle 3.2
Si un seul composant logiciel est disponible pour assurer une fonctionnalité modifiant la sémantique des données
Alors il est instancié par le système.

Règle 3.3
Les composants logiciels ne modifiant pas la sémantique des données sont instanciés par le système.

Règle 3.4
Si un utilisateur souhaite une fonctionnalité ou un composant particulier
Alors il doit doit la (resp. le) spécifier au degré 0 (les degrés 1 et 2 étant optionnels suivant les exigences de l'utilisateur).

Règle 3.5
Si une fonctionnalité correspond à un composant logiciel de configuration ET en entrée
Alors il doit être spécifié par l'utilisateur.

Remarque 3.1 *Lorsque le système instancie un composant logiciel, il lui affecte des caractéristiques de niveau 0, 1, et 2.*

L'utilisation de ces règles permet à l'utilisateur de se concentrer sur les fonctionnalités motrices de son application, le système se chargeant des tâches sous-jacentes en particulier les tâches d'interopérabilité. Le fait de laisser au système non seulement le choix de placement des composants logiciels, mais également la possibilité de sélectionner certains composants logiciels, augmente le nombre de combinaisons possibles de composants en réponse à une requête de l'utilisateur, offrant des perspectives d'optimisation.

3.3.2 Résolution des compositions

Au niveau de la représentation, les composants logiciels sont reliés entre eux selon leur correspondance de types, constituant ce que nous appelons un graphe d'interopérabilité. Ce graphe résout de manière implicite les problèmes de transformation de formats entre deux composants logiciels adjacents dans le graphe.

À chaque nœud et à chaque arc sont associés un ensemble de poids (compris entre 0 et 1) correspondant, entre autres, à la disponibilité des machines ainsi qu'aux distances intersites. Ces poids résultent de l'évaluation effectuées par les Agents Application (voir 3.2.2) : plus le poids est proche de 1, plus l'utilisation de cette ressource est à proscrire.

La recherche d'une composition correspond à la recherche d'un chemin minimisant ces contraintes. Nous utilisons un algorithme d'apprentissage (basé sur les principes du Q_learning [KLM96] et présenté en Annexe C), qui présente l'avantage de tenir compte de l'inexactitude de la représentation (voir 2.2). Basé sur le dilemme exploration/exploitation, il permet l'exploitation des ressources ayant données satisfaction et l'exploration de ressources inconnues ou mal connues. Le parcours du graphe se fait de manière probabiliste en fonction des divers poids des arcs et des nœuds.

3.3.3 Gestion des compositions

Les Agents Solution gèrent un ensemble de compositions structurelles correspondant à une même composition fonctionnelle. À chaque composition est associé un poids (correspondant à la somme des poids des composants évalués par les Agents Application). La composition de plus faible poids est sélectionnée pour le traitement des données de l'utilisateur.

4 Discussion

L'utilisation de capacités d'apprentissage pour la résolution de compositions permet l'adaptation de la représentation à la dynamicité de l'environnement, mais c'est insuffisant dans le cadre d'un environnement très dynamique : le système doit sans cesse apprendre et désapprendre ce qui est coûteux. De plus, l'apprentissage par renforcement ne permet la «capture» de régularités. L'apprentissage des régularités temporelles nécessite l'utilisation d'un niveau indépendant comme nous l'avons réalisé avec la gestion des versions au sein des Agents Solution.

La distribution engendre, dans le cadre de notre travail, une prolifération de processus (Agents Application, Agents Solution), une prolifération du nombre de connexions, ainsi que du nombre des interfaces (pour la spécification des requêtes et pour la spécification des données à traiter). Ces processus et connexions soulèvent le problème de leur rémanence suite à la déconnexion de l'utilisateur ou de l'arrêt des machines.

ADAM provient d'une construction incrémentale fonction d'une recherche d'autonomie et de rapidité croissante. La résolution des conflits entre la centralisation et la distribution, nous a amené à définir plusieurs types d'agents à grains variables :

1. les Agents Application : ils gèrent les communications des données entre le composant logiciel encapsulé et les autres membres de la composition et évaluent la disponibilité

des ressources.

2. les Agents Solution : ils gèrent les communications des données et des résultats avec :
   - l'utilisateur pour l'échange de données et des résultats d'une composition.
   - ADAM pour la gestion des requêtes ainsi que la mise à jour des diverses versions structurelles.
3. le noyau de résolution : il gère les communications entre ADAM et l'utilisateur pour le contrôle de la requête.

Bien que ces trois types d'entités ne soient pas égales en taille, celles-ci contrôlent leur communication de manière équivalente (que faire, quoi faire, quand le faire) ; c'est en cela que nous les appelons «agents». Ces moyens de communication constituent l'unique interface entre l'agent et son environnement.

Les difficultés rencontrées lors de la lors de mise en œuvre de ce système aussi bien au niveau :

- des communications avec les problèmes de synchronisation, de sauvegarde des résultats partiels ainsi que la gestion des compositions hiérarchiques,
- de la spécification de la requête,

justifient la nécessité de la création d'une boîte à outils simplifiant les opérations à effectuer par le concepteur et l'utilisateur.

Ainsi, une boîte à «outils de communication» permettrait :

- la sauvegarde des compositions avec la possibilité de lancement des composants logiciels et la maintenance des liaisons de manière dynamique entre Agents Application au moment de leur utilisation afin d'éviter la prolifération des processus et connexions.
- la gestion des messages mono-application et multi-application avec synchronisation et sauvegarde des résultats partiels.
- la possibilité de contrôle de l'exécution avec reprise en cas de pannes réseau ou d'échec, ce qui pose le problème de la sauvegarde des résultats antérieurs (voir 2.5).

La disponibilité d'une boîte à «outils de création» permettant :

- la spécification simple de composants logiciels, c'est-à-dire spécification de leurs caractéristiques ET de leurs comportements.
- l'utilisation d'un ensemble d'outils prédéfini tels «Insertion», «Migration», etc avec la possibilité pour l'utilisateur de les compléter grâce à une bibliothèque de fonctions (Shell par exemple).
- la gestion des composants logiciels ne pouvant être lancés en batch.

Remerciements

Je remercie vivement Éric Jacopin et Jean-Pierre Briot pour nos fructueuses discussions ainsi que leurs relectures critiques.

Références

[CM96] S. A. Chien and H. B. Mortenson. Automating Image Processing for Scientific Data Analysis of A Large Image Database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1996.

[DGS94] S. Dalmas, M. Gaëtano, and A. Sausse. Distributed Computer Algebra: the Central Control Approach. In *Parallel Symbolic Computation (PASCO)*, 1994.

[DJ94] E. Duxbury and D. Jensen. *VICAR USER'S GUIDE*. http: www-mipl.jpl.nasa.gov/PAG/public/vug/vugfinal.html, October 1994.

[DJR95] T.T. Drashansky, A. Joshi, and J.R. Rice. *SciAgents* - An Agent Based Environment for Distributed, Cooperative Scientific Computing. In *IEEE International Conference on Tools with AI*, 1995.

[DL91] E.H. Durfee and V.R. Lesser. Partial Global Planning : A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transaction on Systems, Man, and Cybernetics*, 21(6):1167–1183, 1991.

[ELS94] O. Etzioni, N. Lesh, and R. Segal. Building Softbots for Unix. In *AAAI Spring Symposium on Software Agents*, March 1994.

[ELST93] O. Etzioni, M.L. Levy, R.B. Segal, and C.A. Thekkath. OS Agents: Using AI Techniques in the Operating System Environment. Technical Report FR-35, University of Washington, dept. of CS & Eng., 1993.

[Fer95] J. Ferber. *Les Systèmes Multi-Agents. Vers une Intelligence Collective*. Inter-Edition, 1995.

[Gir95] M. Girard. An easy way to construct distributed software. In *First International Workshop on Knowledge-Based systems for the (re)use of Program libraries (KbuP'95)*. INRIA, 1995.

[Gir96] M. Girard. Assistant Agents for creation and management of distributed Applications. In *Height IEEE International Conference on Tools with Artificial Intelligence (ICTAI96)*, 1996.

[KLM96] L.P. Kaebling, M.L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kru92] C.W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, 1992.

[LOG97] LOGICAL VISION LTD. WiT. http://www.logicalvision.com:80/index.htm, 1997.

[MH95] D. McDermott and J. Hendler. Planning : What it is, what it could be, an introduction to the special issue on plannig and scheduling. *Artificial Intelligence*, 76:1–16, 1995.

[MLF96] J. Mayfield, Y. Labrou, and T. Finin. Evaluation of KQML as an Agent Communication Language. In J. P. Muller M. Wooldridge and M.Tambe, editors, *Intelligent Agents-Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, volume II. Lecture Notes in Artificial Intelligence, Springer-Verlag, 1996.

[MM94] A. Mili and R. Mittermeir. Storing and retrieving components : A refinment based system. In *16Th. International Conference on Software Engineering.* IEEE Computer Society Press, 1994.

[MMdP95] M. Marcos, S. Moisan, and A.P. del Polil. Verification and Validation of Knowledge-Based Program Supervision Systems. In *IEEE International Conference on Systems, Man, and Cybernetics.* IEEE Edition, October 1995.

[Mue96] H.J. Mueller. Negociation Principles. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 211–230. Wiley Interscience, 1996.

[Pit90] J. Pitrat. *Métaconnaissances. Futur de l'intelligence Artificielle* . Hèrmes, 1990.

[Rif93] J.M. Rifflet. *La Programmation sous Unix.* Ediscience International, 1993.

[Smi80] R.G. Smith. The contract net protocol : highlevel communication and control in a distriuted problem solver. *IEEE Transaction Computer*, 29:1104–1113, 1980.

[vdEvHT95] J. van den Elst, F. van Harmelen, and M. Thonnat. Modelling Software Components for Reuse. In *Seventh International Conference on Software Engineering and Knowledge Engineering*, pages 350–357. Knowledge Systems Institute, June 1995.

[VMT96] R. Vincent, S. Moisan, and M. Thonnat. Une bibliothèque pour des moteurs de pilotage des programmes. Technical Report 3011, INRIA-Sophia Antipolis, Octobre 1996.

1 Annexe A : Les dépendances fonctionnelles

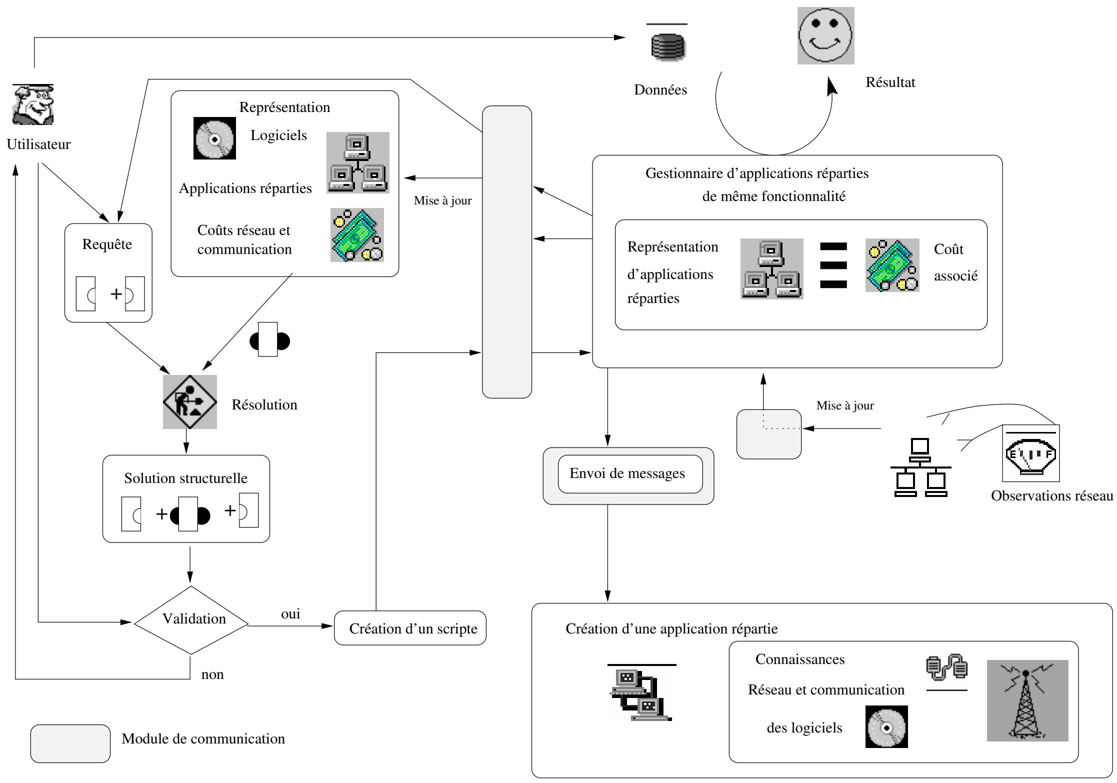

FIG. 5 - *Les dépendances fonctionnelles*

La requête de l'utilisateur est complétée par ADAM lors d'une phase de résolution dont l'algorithme est présenté dans l'annexe C. La composition structurelle obtenue, si validée par l'utilisateur, engendre la création d'un Agent Solution responsable de

- la création de l'application répartie à partir d'un script reçu.
- la gestion des versions d'applications réparties ainsi que de la gestion de la représentation de leur coût.
- l'envoi des données de l'utilisateur vers l'application répartie, puis retour du (ou des) résultat(s) obtenu(s) vers ce dernier.

La création de l'application répartie se fait par l'intermédiaire d'un script envoyé par le noyau central à l'Agent Solution crée, il est ensuite transformé en un ensemble de messages transmis au réseau : ces messages servent à activer les agents Application impliqués dans la chaîne solution et leur spécifient les connexions à créer ou à ouvrir. La création des connexions entre Agents Application nécessitent des connaissances réseau et communication réparties au sein des Agents Application.

La mise à jour de la représentation des coûts des applications réparties se fait par l'envoi de messages à l'attention de l'Agent Solution par l'intermédiaire de canaux de communication.

2 Annexe B : Les différentes natures de contrôle

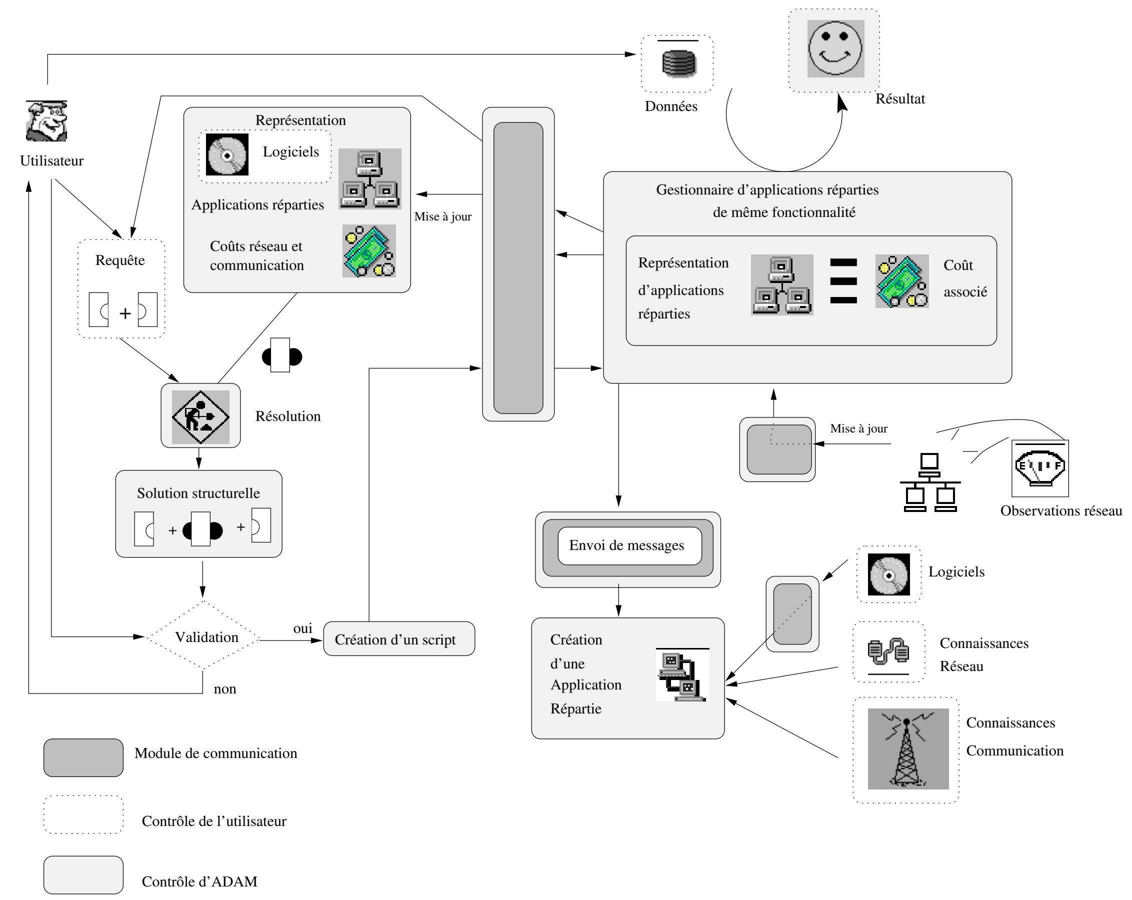

FIG. 6 - *Les différents natures de contrôle*

L'utilisateur contrôle les entrées et les sorties du système ADAM. En effet, il possède les connaissances nécessaires à la spécification d'une requête cohérente et doit ensuite valider le résultat retourné par le système. Il contrôle de la même manière les données devant être traitées par l'application répartie et juge de la cohérence du résultat retourné.

Afin d'augmenter l'autonomie du système, ADAM met automatiquement à jour sa représentation des applications réparties ainsi que leur coût associé au niveau du noyau et des Agents Solution.

3 Annexe C : L'algorithme de résolution

Nous proposons un algorithme de résolution permettant la recherche d'un chemin dans un graphe composé de noeuds et d'arcs. Les noeuds représentent les composants logiciels placés sur des machines spécifiques et les arcs, le réseau. Le requête de l'utilisateur spécifie un ensemble de noeuds et l'objectif de l'algorithme est de trouver un chemin passant par ces derniers minimisant un critère de coût. Les noeuds spécifiés par l'utilisateur définissent des sous-problèmes à résoudre, chacun d'eux étant caractérisé par un noeud de départ et un noeud d'arrivée que nous appelons noeud_start et noeud_end.

L'algorithme de résolution que nous proposons est similaire au Q_learning [KLM96] et permet *pour un noeud d'arrivée donné* :

– de mémoriser les réussites et les échecs lors de la recherche d'une solution.

– de quantifier la qualité d'une solution.

– d'explorer de nouvelles solutions.

Nous appelons «solution» tout chemin menant au but, c'est-à-dire au noeud_end.

3.1 Les différents poids

Soit un graphe composé d'arcs et de noeuds. À chaque noeud i est associé un poids W_i représentant les contraintes liées au choix de ce composant et à chaque arc trois poids C_i, P_i et PB_i. W_i est le résultat des tests de charge des machines (voir section 3.3.2), C_i correspond au coût des communications réseaux intersites et les deux poids P_i et PB_i sont spécifiques à la recherche et peuvent être considérés comme une pénalité pour les arcs ne menant pas au but. Ces poids prennent des valeurs comprises entre 0 et 1 : plus la valeur est proche de 1, plus le choix de l'arc et du noeud est à proscrire. Les évaluations des tests effectuées par ADAM pour la mise à jour des poids W_i et C_i retournent des temps qui doivent être convertit comme nous l'indiquons dans la phase d'initialisation de l'algorithme (voir section 3.4).

3.2 La règle de sélection d'action

À chaque noeud n, le choix d'un couple $(arc; noeud)$ est effectué selon une règle d'action basée sur un choix aléatoire, proportionnel aux poids des couples $(arc; noeud)$ successeurs du noeud n. La probabilité pour que ns *n*oeud *s*uccesseur du noeud n via l'arc n-ns soit sélectionné est la suivante :

$$Proba(n, n-ns, ns) = \frac{(W_{ns} + C_{ns} + P_{n-ns}) * PB(n-ns)}{\sum_{i \in succ} (W_i + C_i + P_{n-i}) * PB(n-i)}$$

3.3 Les règles de récompense

Afin de mémoriser une trace des réussites et des échecs lors de la recherche d'un but déterminé, les différents poids W_n, P_n et PB_n obtiennent une *récompense* récursivement du noeud de terminaison (qui peut être le noeud_end spécifié ou un noeud sans successeur) au noeud de départ selon les règles suivantes :

REMARQUE 3.1 *Nous utilisons les définissions suivantes:*

graphe : *ensemble des noeuds du graphe.*
noeud_start : *noeud de départ.*
noeud_end : *noeud d'arrivée.*
noeud_de_terminaison : *noeud_end ou noeud sans successeur.*
chemin : $\{noeuds\} \supset \{noeud\_start; noeud\_de\_terminaison\}$.
solution : $\{noeuds\} \supset \{noeud\_start; noeud\_end\}$.
n : $\forall n \in \{chemin\}$.

RÈGLE 3.1

```
If n ∈ {solution} Do
    update(W_n);          /* modifications selon la règle 3.2 */
    P_n := 0.0;           /* réinitialisation des pénalités de recherche */
    PB_n := 1.0;
End If
```

RÈGLE 3.2

```
If n ∈ {solution} Do
    If n = noeud_end Do
        W_n := R_n;           /* R_n étant le poids représentant la contrainte *
                              de charge réelle du noeud n.*/
    Else
        /* n ≠ noeud_end */
        W_n := R_n * W_s;     /* s étant le successeur sélectionné selon la règle d'action */
    End If
End If
```

RÈGLE 3.3

```
If n ∉ {solution} Do
    P_n := ((W_n + C_n) + (W_s + C_s)) * TAUX_PÉNALITÉ
    /* s étant le successeur de n */
    /* on augmente la pénalité de l'arc correspondant à l'arc(n-s) */
End If
```

RÈGLE 3.4

```
If exploré(succ(n)) = TRUE Do      /* tous les successeurs du noeud n ont été explorés */
    PB_n := 0.1;                   /* on coupe l'accès aux branches en sortie du noeud */
End If
```

On remarque que W_n représente soit :

- la contrainte de charge mesurée à l'aide des tests de charge et de performance que nous appelons (R),
- soit la qualité de la portion de chaîne descendante par multiplication des représentations successives R des noeuds impliqués dans celle-ci.

3.4 Algorithme de recherche

1. *Phase d'initialisation*

```
    #DEFINE MAX_100 100
    For each noeud i ∈ GRAPHE
    Do
        W_i  := (MAX_100 - temps_resultat_test)/100;
        C_i  := (MAX_100 - temps_distances)/100;
        P_i  := 0.0;
        PB_i := 1.0;
    End For
```

2. *Phase de recherche d'un chemin*

/* Lors de cette phase d'activation, un chemin est crée (une liste */
/* de noeuds) partant du *noeud_start* spécifié par l'utilisateur et se */
/* terminant à un noeud de terminaison qui peut être le noeud */
/* d'arrivée souhaité (*noeud_end*) ou bien une branche sans feuille. */

```
    chemin := ∅
    solution := ∅
    noeud_courant := noeud_start;
    While ((noeud_courant ≠ noeud_end) or (succ(noeud_courant)) ≠ ∅)
    Do
        Liste_Proba := ∅;
        For each i ∈ succ(noeud_courant) Do
            Liste_Proba := calcul_proba(i) ∪ Liste_Proba;
        End For
        noeud_selectionne := choix_d_un_succ(Liste_Proba);
        chemin := {chemin} ∪ noeud_selectionne;
        noeud_courant := noeud_selectionne;
    End While
    If noeud_terminaison = noeud_end
    Do
        solution := chemin;
    End If
```

3. *Phase de récompense*

```
    For i := noeud_terminaison to noeud_start
    Do
        récompense(W_i,C_i,P_i,PB_i, solution);
    End For
```