MadKit: une architecture de plate-forme multi-agent générique

Olivier Gutknecht, Jacques Ferber, Fabien Michel

▶ **To cite this version:**

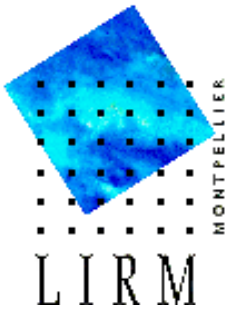
Olivier Gutknecht, Jacques Ferber, Fabien Michel. MadKit: une architecture de plate-forme multi-agent générique. [Rapport de recherche] R.R.LIRMM 00061, Lirmm, University of Montpellier. 2000. hal-02404016

HAL Id: hal-02404016
https://hal.umontpellier.fr/hal-02404016

Submitted on 11 Dec 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



LABORATOIRE D'INFORMATIQUE, DE ROBOTIQUE
ET DE MICROÉLECTRONIQUE DE MONTPELLIER

Unité Mixte CNRS - Université Montpellier II C 09928

RAPPORT DE RECHERCHE

MadKit: une architecture de plate-forme multi-agent générique

Olivier GUTKNECHT
gutkneco@lirmm.fr
Fabien MICHEL
fmichel@lirmm.fr

Jacques FERBER
ferber@lirmm.fr

05/2000 R.R.LIRMM 00061

161, Rue Ada - 34392 Montpellier Cedex 5 - France
Tél: (33) 67 41 85 85 - Fax: (33) 67 41 85 00

Résumé

Dans ce rapport, nous présentons l'architecture de MadKit (pour "Multi-Agent Development Kit"), une plate-forme générique de conception et d'exécution de systèmes multi-agents. Cette plate-forme a l'originalité d'être basé sur un modèle organisationnel plutôt qu'une architecture d'agent ou un modèle d'interaction spécifique. L'utilisation de groupes et de roles associés à des agents est mis en oeuvre tant en tant qu'outil de modèlisation et de conception pour les développeurs de systèmes multi-agents, que de principe d'architecture de la plate-forme elle-même.

Cette architecture est basée sur un noyau agent minimal découplé de tout modèle individuel d'agent. Dans cette plate-forme, les services classiques de passage de message distribués, de migration ou de surveillance sont fournis au meta-niveau par des agents spécialisés afin d'obtenir un maximum de flexibilité. Une interface graphique componentielle et découplée du noyau et des agents permets de supporter différentes modes d'utilisation et d'exploitation de la plate-forme.

Nous illustrons notre propos en présentant certaines conséquences et déclinaisons de cette plateforme, ainsi que que quelques applications construites avec MADKIT.

1 Introduction

1.1 Vers une programmation multi-agents ?

Une partie importante des réflexions actuelles dans le monde des systèmes multi-agents a trait à la mise en application de ces modèles. Deux problèmes majeurs sont les questions des méthodologies de conception et des supports d'exécution : est-il possible de penser et implémenter un système par une programmation multi-agents ?

Ces deux points posent en fait la même question : existe-t'il des modèles et outils *spécifiques* à une approche multi-agents ? les modèles et technologies existantes (UML, CORBA, ...) - principalement venues du monde des objets[1] - sont-elles suffisement expressives et adaptée à un point de vue agent ?

Notre hypothèse de travail est qu'il existe effectivement des modèles et techniques à construire pour l'*implémentation* de tels systèmes. Nous aborderons dans ce rapport la question du support d'exécution.

Si nous faisons un rapide tour d'horizon, nous remarquons d'abord que la plupart des plate-formes agents sont conçues par rapport à un modèle d'agent particulier (JAT-lite [9], Zeus [15], JAF [13], Sodabot [4]), ou un domaine d'application (la simulation pour Swarm ou CORMAS, les agents mobiles pour Aglets ou Odyssey [10]). On peut également noter la disparition des plupart des plate-formes basées sur des langages spécifiques (tel que le fut Telescript) pour des bibliothèques associés à des langages objets classiques.

Un autre domaine actif dans le monde des plate-formes est celui des implémentations conformes aux spécifications FIPA [7]. L'inconvénient de cette approche est qu'elle sous-entend un modèle d'agent très particulier. Il faut en effet être capable de mettre en oeuvre la communication par actes de langages définie par ACL, dans l'espoir d'avoir une base formelle et validable (ce que les premières expérimentation semblent démentir, voir par exemple [16]). Cela exclut d'emblée la majorité des approches émergentistes. De plus, d'autres pré-requis structurent fortement le style d'implantation des plate-forme (comme le choix de CORBA comme mécanisme de liaison).

[1] et dans une moindre mesure de la communauté commonKADS

En particulier, nous désirons mettre en avant l'idée que l'interopérabilité dans les systèmes multi-agents devrait être conçue *à un niveau de description "agent"*. Les solutions d'intégration actuelles dans les architectures logicielles telles que XML ou CORBA sont intéressantes par les fondations fortement structurées qu'elles procurent, mais ne sont pas à notre avis la solution universelle à nos préoccupations à cause du décalage entre les différents niveaux d'abstraction concernés (description, objet, agents). Tout au moins, la relation entre ces technologies et le niveau d'expression agent devrait être clairement identifiée et définie.

1.2 MadKit comme multi-système-multi-agent

Nous avons construit un modèle, nommé AALAADIN qui tente de structurer des systèmes multi-agents en prenant un point de vue principalement organisationnel. Nous avons par le passé présenté divers aspects de ce modèle. Néanmoins, nous n'avons abordé jusqu'à maintenant que par la bande la question du support d'implémentation dans ces travaux. Dans ce rapport, nous allons nous concentrer sur la présentation de l'architecture de la plate-forme SMA MADKIT et des conséquences.

Nous verrons que l'utilisation de structures organisationnelles permet de garder une cohérence globale et facilite l'intégration et l'exécution simultanée d'agents hautement hétérogènes dans une même plate-forme, d'où l'idée de "multi-systèmes-multi-agents". MadKit, en tant qu'outil de développement de SMA, a été motivé par le besoin d'avoir une plate-forme la plus souple possible, et capable de s'adapter au différents modèles d'agents et domaines d'applications, - fort nombreux - du domaine. De la même manière, nous avions pour objectif de pouvoir remplacer aisément l'implémentation des fonctionalités de base de la plateforme selon les besoins.

ce rapport est structuré comme suit. Nous reviendrons brièvement sur le modèle AALAADIN dans la seconde section de ce rapport. La troisième partie décrit l'architecture de base de la plate-forme, avec en particulier la notion de "micro-noyau agent". La section 4 décrit les déclinaisons possibles, tant au niveau des modèles d'agents que des services de base agentifiés ou des applications. La cinquième section présente quelques expérimentations, et nous terminons par quelques perspectives.

2 Modèle organisationnel

Nous présentons ici très rapidement le modèle de description organisationnel qui structure la plate-forme MadKit. Ce modèle a été décrit en détail dans [5]. Sa sémantique opérationnelle basée sur le π-calcul a été proposée dans [6]. Quelques perspectives méthodologiques ont été abordées dans [11].

2.1 Agent

Quasiment aucune contraite n'est posée sur l'architecture interne ou le modèle de comportement de l'agent. L'*agent* est simplement décrit comme une entité autonome communiquante qui joue des *rôles* au sein de différents *groupes*. La très faible sémantique associée à l'agent dans ce modèle est tout à fait volontaire. L'optique est bien de laisser la définition d'agent en retrait, non seulement pour ne pas prendre part dans le débat classique de "qu'est-ce qu'un agent", mais surtout pour laisser au concepteur toute liberté pour choisir l'architecture interne appropriée à son domaine applicatif. Cette liberté se retrouvera dans les principes de MadKit

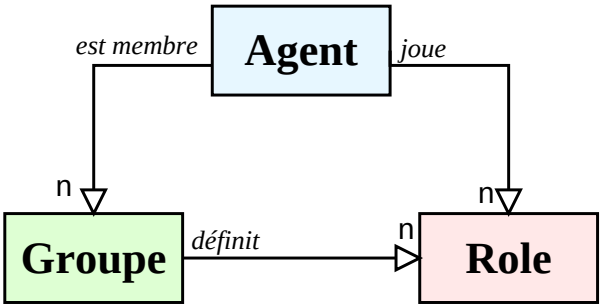


FIG. 1 – Modèle organisationnel

2.2 Groupe

Le *groupe* est la notion primitive de regroupement d'agents. Chaque agent peut être membre d'un ou plusieurs groupes. D'une façon un peu simpliste, un groupe peut être vu comme un moyen d'identifier par regroupement un ensemble d'agents. Plus classiquement, associé au rôle, il définira la structuration organisationnelle d'un SMA usuel. Les différents groupes peuvent se recouper librement.

2.3 Rôle

Le rôle est une représentation abstraite d'une fonction, d'un service ou d'une identification d'un agent au sein d'un groupe particulier. Chaque agent peut avoir plusieurs rôles, un même rôle peut être tenu par plusieurs agents, et les rôles sont locaux aux groupes. L'hétérogénéité des situations d'interaction est rendue possible par le fait qu'un agent peut avoir plusieurs rôles distincts au sein de plusieurs groupes, les communications étant cloturées par les groupes.

3 Architecture

3.1 Principes

La structure organisationnelle que nous venons de décrire est implémentée au coeur de la plate-forme MADKIT, tant pour fournir un modèle organisationnel aux SMA exécutés que pour le fonctionnement interne du système. L'utilisation de groupes d'agents pour la structuration de plate-formes a été proposée dans d'autres architectures, comme par exemple [1], bien que ces mécanismes sont limités aux problèmes de migration et ne possèdent pas ce trait distinctif de prise en compte de groupes et rôles simultanés pour un agent donné.

En plus de ce modèle d'organisation, MadKit est basé sur trois principes :

– Architecture à micro-noyau
– Agentification systématique des services
– Découplage applicatif entre noyau, agents et application d'accueil.

Concrètement, MadKit est un ensemble des packages Java qui implémentent le noyau agent, diverses bibliothèques de base de messages, d'agents et de sondes. Le choix de Java a été fait très tôt pour des raisons de portabilité, mais également pour

la richesse des bibliothèques de bases, les possibilités de sérialisation d'objets et de présence de réflexivité, même sommaire.

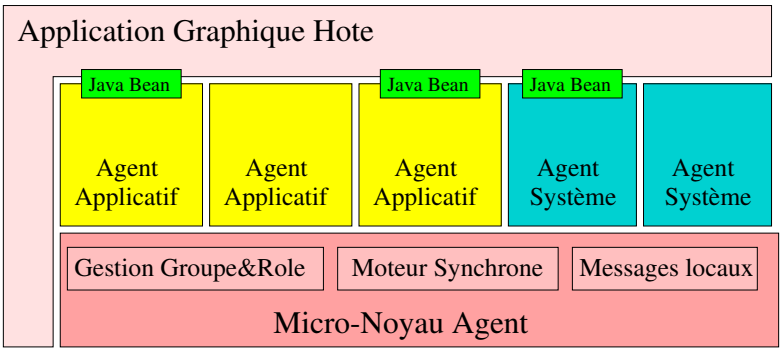


FIG. 2 – Architecture générale

Le principe essentiel de MADKIT est d'utiliser partout où cela est possible la plate-forme pour sa propre gestion et exécution. Tous les services hors ceux assurés par le micro-noyau sont implémentés par des agents à part entière. Ceci vient à la fois d'un souci d'unicité de modèle, mais également de mise à l'épreuve des SMA comme modèle de programmation. Cela le rapproche également fortement des approches meta en architectures objets [20] à ceci près que nous plaçons la relation au meta entre agents applications et "systèmes" (communication, ordonnancement, surveillance...).

MadKit n'est pas une plate-forme agent dans le sens classique, centrée autour d'un modèle particulier d'agent ou d'interaction. La taille réduite et les fonctionnalités du noyau de base, la neutralité des types agents, associée à ce principe de services agentifiés et de découplage par rapport aux applications "hôtes" permet en fait d'obtenir toute une gamme d'environnements d'exécutions aux finalités parfois complètement opposées.

3.2 Le micro-noyau agent

Le "micro-noyau" de MADKIT est un environnement d'exécution d'agents de taille réduite (moins de 40 Ko). Le terme "micro-noyau" est utilisé ici intentionnellement comme référence au modèle des systèmes d'exploitation à micro-noyau. On pourrait directement plagier leur principe fondateur [17] '*incorporating in micro-kernels a small number of key facilities that allow the efficient deployment of system services.*" en remplaçant "system" par "agents".

3.2.1 Fonctionnalités

Le noyau ne prend en charge que les fonctions suivantes :

Gestion des groupes et rôles locaux Comme l'interopérabilité et les mécanismes d'extension de MadKit se basent sur le modèle agent-groupe-rôle, il est essentiel que cette information organisationnelle soit gérée au plus bas niveau afin que tous les agents, quels que soient leurs modèles individuels, y aient accès. Le noyau a la responsabilité de maintenir une information correcte sur les membres des

groupes et les rôles tenus. Il vérifie également si les requêtes faites sur le système de groupes et rôles sont acceptables (c'est à dire en évaluant ou délégant les fonctions de rôles).

Gestion du cycle de vie des agents Le noyau gère également le lancement (et éventuellement l'arret) des agents et maintient les tables de références sur les objets d'implémentation. Il est également le gestionnaire des informations administratives sur les agents (possesseur, modalités de créations, ...) et donne un identifiant garanti unique à chaque agent. Cet identifiant, l'`AgentAddress`, est forgé à partir de l'addresse du noyau MadKit et l'identification de l'agent sur le noyau ; il peut être rapproché de la notion de GUID de FIPA. La forme de cet identifiant peut également être redéfinie pour faciliter l'intégration avec d'autres plateformes agent.

Passage de message local Le noyau a la responsabilité de l'aiguillage et de la distributions de messages entre agents **locaux** (s'exécutant sur le noyau). Le passage de message au plus bas niveau revient à de simples échanges de références pour pouvoir facilement implémenter différentes sémantiques de passage de message à un niveau supérieur.

Le noyau lui-même est en fait encapsulé dans un agent particulier, le `KernelAgent` qui est créé automatiquement à l'amorçage de la plate-forme. Il agit comme représentant du noyau dans le monde agent : toutes les demande de surveillance ou de contrôle sont alors écrites comme une interaction inter-agent et préservent l'unicité du modèle

3.2.2 Mécanismes de passage au meta-niveau

Le noyau lui-même est extensible par des "hooks", des points d'interceptions sur ses opérations. Un agent autorisé (par exemple en ayant rempli les conditions d'accès au groupe "système") a la possibilité de demander un accès à l'un de ces hook. La demande se fait par une interaction agent avec le `KernelAgent` mentionné plus haut.

Ce hook sont un mécanisme de publication/abonnement qui permettent soit la surveillance, soit l'interception des opérations. Quasiment toutes les fonctionnalités de base du noyau (lancement d'un agent, accès et modifications à la structure organisationnelle, passage de message) sont concernées. Cela permet donc d'écrire facilement des agents d'analyse d'un SMA en fonctionnement (dans le style de [19]), ou bien des agents étendant les fonctionalités de base de la plate-forme.

Le nombre réduit de fonctionalités bien définies favorise la modification du comportement des opérations de base tout en préservant au maximum leur sémantique. Pour toute opération noyau, il existe deux types de hooks :

Les hooks de surveillance Un nombre quelconque d'agents peut s'abonner un tel hook. Dans ce cas, toute invocation de l'opération concernée provoquera l'envoi d'un message d'information comprenant le contexte de l'action par le `KernelAgent` vers les abonnés. Par exemple, on peut ainsi faire tracer par un agent les messages échangés dans un SMA en espionnant l'opération de passage de message.

Les hooks d'interception Dans ce cas, un seul agent peut utiliser un hook d'interception sur une opération du noyau, selon un pattern de délégation. L'agent reçoit alors le contexte qui aurait dû être celui de l'opération, laquelle n'ayant donc pas lieu : l'agent peut alors définir un nouveau comportement. Pour continuer sur l'exemple de l'opération de passage de message, les agents qui permettent

à MadKit de fonctionner en architecture distribuée interceptent ces invocation pour pouvoir router les messages non-locaux via un protocole réseau.

L'autre mécanisme d'extension du noyau consiste à invoquer manuellement des opérations de base. Dans ce cas, ce sont des agents habilités qui enverront une demande au KernelAgent pour que le noyau effectue une opération. Par exemple, un agent de communication réinjectera par ce biais les messages venus d'une machine distante dans le noyau local.

3.3 Structure et fonctions d'un agent

La classe de base d'un agent MadKit (`AbstractAgent`), définit quelques fonctionnalités de base pouvant être nécessaires dans les modèles classiques.

3.3.1 Fonctionnalités

Les fonctions associées à tout agent sont :

Cycle de vie L'agent dispose de quatre états (création, activation, exécution, et destruction), et a la possibilité de démarrer d'autres agents sur le noyau local (et les désactiver par la suite). Par contre, aucun mécanisme concret d'exécution n'est défini à ce niveau.

Communication La communication est implémentée sous forme de passage de message asynchrone, soit d'agent à agent identifiés par leur `AgentAddress` ou leur groupe et rôle, soit sous la forme d'une diffusion à tous les teneurs d'un rôle dans un groupe donné.

Organisation Tout agent dispose de primitives permettant d'observer son organisation locale (connaître les groupes et rôles courants) et d'y agir (prise de rôle, entrée et retraits de groupes).

Outils La classe de base des agents permet également de manipuler une éventuelle interface graphique associée à l'agent, les flots d'entrée/sortie, etc.

3.3.2 Messages

Les messages sont définis par héritage à partir d'une classe de base qui ne définit que la notion d'émetteur et destinataires. Certain messages de base sont néanmoins fournis (encapsulation d'une chaîne ou d'un objet arbitraire, messages KQML et FIPA-ACL, messages XML), mais restent extensibles pour s'adapter à tout protocole d'interaction.

3.3.3 Threads et moteur synchrone

Pour faciliter les implémentations de modèles classique d'agents autonomes, cette classe de base est déclinée vers une implémentation associant un thread d'exécution à un agent.

Néanmoins, pour certains types de systèmes, en particulier les SMA réactifs, il est nécessaire de gérer un grand nombre d'agents (jusqu'à centaines de milliers) de granularité assez fine, et de contrôler leur ordonnancement. Cela est quasi-impossible si l'on se repose sur un mécanisme de thread à cause de la surcharge induite. Dans ce cas, on utilise ces agents "synchrones" via des agents externes qui vont définir leur politique d'exécution synchrone. Des agents d'observation peuvent être ajoutés pour

avoir une vue globale sur certaines propriétés, à l'instar des plateformes classique de simulation comme CORMAS [3] ou Swarm [14].

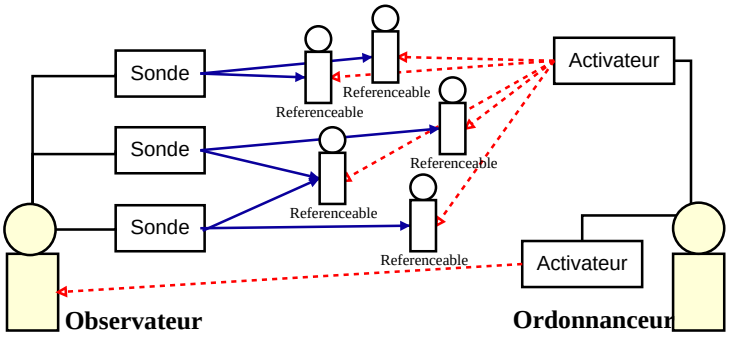


FIG. 3 – Fonctionnement des agents synchrones

Ce modèle permet l'utilisation de plusieurs ordonnanceurs simultanés en structurant un système simulé par le mécanisme de groupe et rôle. Il est décrit plus complètement dans [12]. L'intérêt de cette implémentation est que les agents exécutés dans ces systèmes synchrones ont exactement les *mêmes fonctionnalités* (passage de message, vue organisationnelle, ...) que les agents threadés, et peuvent donc être associés facilement en systèmes hybrides. De plus, la gestion du mécanisme de simulation ou d'observation repose encore une fois sur l'approche en groupe et rôle (on va par exemple associer une sonde à une certaine propriété présente dans les tenants d'un rôle donné sur un certain groupe).

4 Déclinaisons

4.1 Modèles d'agents

La définition d'un agent étant plutôt minimaliste, différentes bibliothèque implémentant des architectures d'agents classiques ont été construites en complément de cette architecture. On peut par exemple citer :

- Une bibliothèque permettant d'implémenter les agents en Scheme [2]
- Une implémentation d'agents faisant une liaison avec le moteur de règles CLIPS de JESS [8], développée conjointement avec l'Université d'Aix/Marseilles. Les propriétés des agents MadKit (par exemple l'organisation) sont traduites automatiquement sous forme d'assertions dans la base.
- Différentes bibliothèques pour des SMA réactifs, comme par exemple un "TURTLE KIT" qui clone une partie des fonctionnalités de l'environnement StarLogo [18].
- Des outils de constructions graphique d'agents, permettant de simuler et d'exécuter en place les designs dans la plateforme (que nous décrivons plus en détail dans la section suivante).
- L'implémentation d'un modèle d'acteurs.

Notons que l'utilisation d'un de ces modèles n'est nullement exclusif : il est courant d'exécuter simultanément sur la même plate-forme MadKit des agents Scheme, réactifs, systèmes, BDI ...

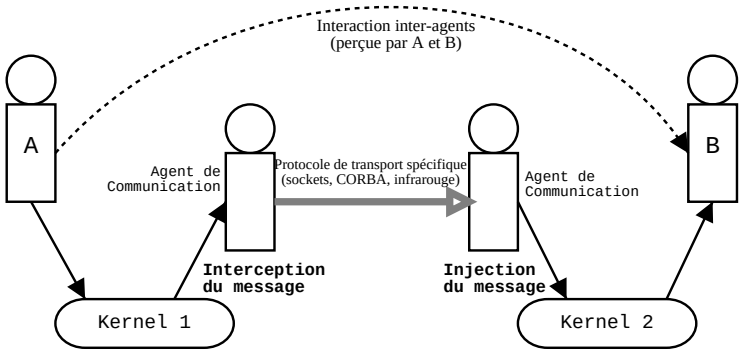


FIG. 4 – Agents de communication distribuée

4.2 Agentification des services

A l'opposé de plateformes plus monolithiques, MadKit utilise donc des agents pour implémenter au niveau meta des services comme le passage de message distribué, la migration d'agent, la sécurité d'organisations d'agents et divers aspects de gestion du système ; et ce éventuellement à l'aide des mécanismes d'extension du noyau.

Comme ces services sont mis en oeuvre par des agents, et décrits par la structure organisationnelle, les interactions entre ces agents systèmes, le noyau, et les agents de l'applications sont décrits dans le modèle agent-groupe-role. Ceci implique qu'un agent offrant une fonctionnalité donnée peut être remplacé par un autre de façon transparente (et éventuellement durant le fonctionnement d'une application), à partir du moment où les groupes, rôles et interactions sont respectées.

Par exemple, des développeurs extérieurs ont échangé nos agents de synchronisation de l'information organisationnelle par d'autres qui fournissaient une interface avec des annuaires distribués. Les agents du SMA applicatif et les autres agents "système" n'ont pas eu à être modifiés et n'ont en fait pas remarqué l'échange.

Baser les services sur des agents décrits en terme de rôles a également l'effet de faciliter une montée en puissance. Un agent tenant plusieurs rôles peut, au fur et à mesure que les besoins de l'application grandissent, déléguer dynamiquement à de nouveaux agents certains de ses rôles pour réduire sa charge.

De plus, la structuration en groupe agissant souvent en "masquage" et délégation d'un SMA complexe, un rôle ne correspond pas forcément à une mise en oeuvre par un et un seul agent. Par exemple, un agent fournissant un rôle de communication système peut être un simple *représentant* d'un groupe d'agents spécialisés prenant en charge chacun un protocole.

Avoir découplé les services du noyau permet aussi de ne pas alourdir les applications : deux plateformes exécutant un SMA localement ou en distribué ne diffèrent que par le lancement ou non des agents de communication (figure 4).

4.3 Applications hôtes

Le noyau de la plateforme ne fournit aucune interface graphique pour l'utilisateur. Il est par contre possible de lui associer une application hôte qui la mettra en oeuvre.

De plus, chaque agent peut définir un objet graphique pour sa représentation, d'un simple bouton à une application classique encapsulée. L'hôte aura alors la charge, au lancement d'un agent, de décider de la mise en place de l'objet graphique défini par l'agent (dans des fenêtres séparées, combiné avec l'interface d'un autre agents, etc..) et de les gérer au niveau global.

Associé au principe d'agentification des services, cela permet de modulariser complètement la plateforme. Par exemple, les variétés suivantes de MadKit ont été mise en oeuvre :

- La G-Box, un environnement graphique complet de développement et de test de systèmes multi-agents. Elle permet de contrôler le cycle de vie des agents, charger des SMA dynamiquement, et de manipuler graphiquement les accointances ou bien les tables de groupes et de rôles.
- Un simple "booter" pour démarrer uniquement une liste d'agents donnés, qui peut être utilisé pour la distribution finale d'une application agent ou le mise en place d'agents (facilitateurs, annuaires, interprètes) sur un serveur.
- Une applet qui empaquète le noyau agent et un SMA applicatif, pour l'exécuter dans un navigateur distant.
- Une application qui embarquerait un noyau MadKit est des agents applicatifs pour prendre en charge les aspects dynamiques ou coopératifs, indistinguable d'une application "classique"
- Pour valider notre approche, nous également avons développé une version de MadKit utilisant le même noyau que la plate-forme complète, et capable de tourner sur des assistants personnels de type Palm. Un SMA applicatif réduit (gestionnaire de rendez-vous) a été implémenté de façon classique et mis en place avec un agent de communication spécifique (par infra-rouge). Le prototype complet fonctionne malgré les limitations fortes en puissance de calcul et occupation mémoire (moins de 64 Ko disponible pour l'application).

5 Applications

5.1 SEdit, un éditeur graphique multi-formalisme

SEdit (pour "Structure Editor") a pour base un éditeur graphique de formalismes représentables sous formes de graphes (UML, réseaux de pétri, diagrammes d'interaction, ...). Il supporte également les structures imbriquées (par exemple d'un paquetage à un diagramme de classes) qui seront ouvertes et liées d'un éditeur à l'autre. Les formalismes sont définis de façon externe, sous forme d'une description XML associé à des classes supplémentaires si l'on veut contrôler finement son apparence ou son comportement (par exemple pour permettre la simulation d'un réseau de Pétri).

Du point de vue de MadKit, cet outil a été construit sous forme d'un ensemble d'agents. Chaque éditeur est en fait un agent MadKit, configuré pour un formalisme donné, et les interactions entre éditeurs (par exemple pour l'imbrication de structures) sont des interactions inter-agents classiques.

Ce système aurait pu être développé sous forme d'une application standard. Nous avons choisi de l'implémenter sous forme d'agents pour les raisons suivantes :

Gain de fonctionnalités via des agents externes Par exemple, l'utilisation conjointe des agents SEdit et des agents de communication permet de faire fonctionner directement l'application en mode distribué, par exemple pour du travail collaboratif sur un modèle multi-formalisme.

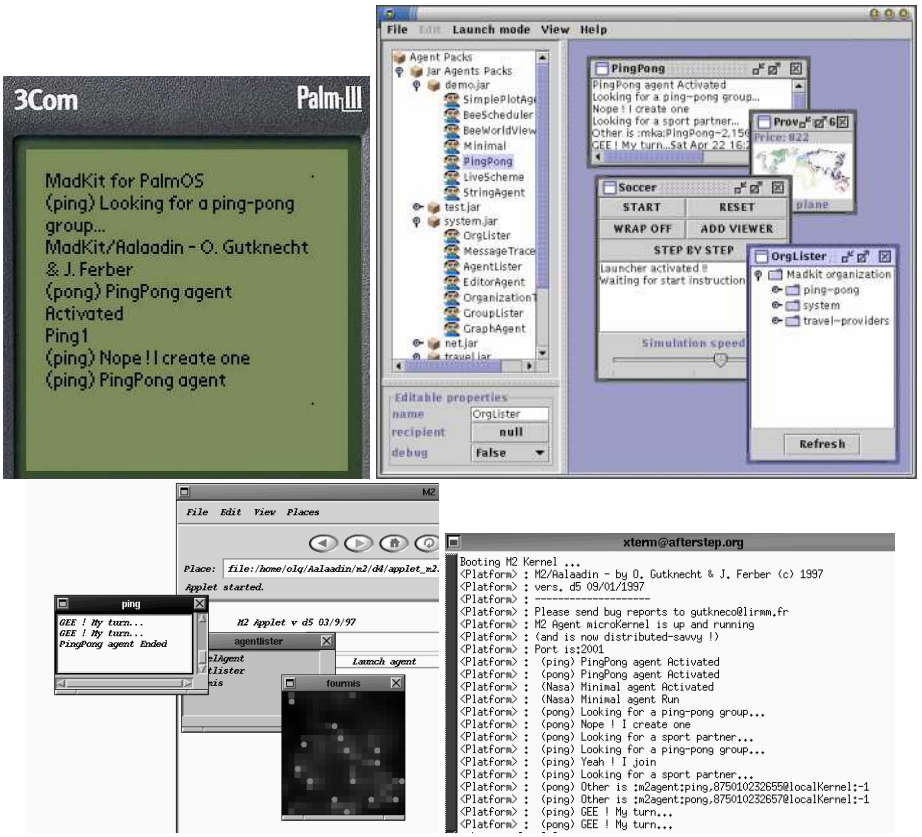


FIG. 5 – Des applications MADKIT dans les hôtes G-Box, console, Applet et Palm

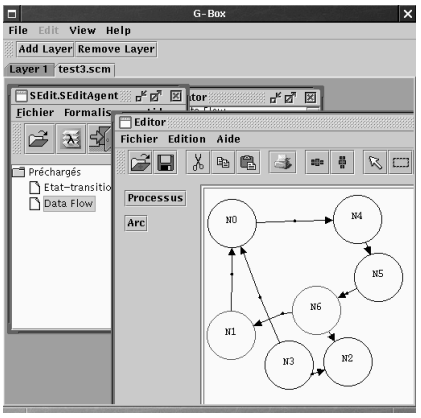


FIG. 6 – L'application SEDIT écrite sous forme d'agents MADKIT

Variabilité d'usage SEdit existe en deux versions : l'une a l'apparence d'une application classique (pour faciliter l'usage de formalismes non spécifiquement agents), l'autre fonctionne à l'intérieur de la G-Box. Les agents SEdit n'ont pas à être modifiés dans les différentes applications hôtes.

Définition d'agent "en place" Certains formalismes associés (réseaux de Pétri, BRIC, règles d'actions) permettent de définir des comportement d'agents et de les simuler. Plutôt que de générer le code des agents et les réintégrer dans MadKit, on peut également tester les agents *en place*, par exemple via certains noeuds du formalisme capturant les messages arrivant à l'éditeur et les réintégrant dans la structure sous forme de jetons. L'agent SEdit joue alors à la fois le rôle d'un éditeur dans le système multi-agents SEdit et le rôle de l'agent applicatif en train d'être construit, et plongé dans son groupe final, sans que ses accointances s'aperçoivent qu'il s'agit d'un agent en construction.

5.2 Autres applications

MadKit est utilisé dans et hors de l'équipe multi-agents du LIRMM dans divers contextes depuis plus de deux ans. Parmi les applications développées, on peu citer : un simulateur d'architectures hybrides pour le contrôle de robots sous-marins à la DGA, l'évaluation de SMA pour le contrôle de chaînes de production, la résolution de problèmes multi-robots, l'administration distribuée de réseaux, et un système de gestion de la connaissance dans les groupes de travail réalisé par la société Euriware.

6 Conclusion

Dans ce rapport, nous avons présenté une plate-forme multi-agent basée sur un modèle organisationnel, et défendu l'idée qu'il est possible d'intégrer au sein d'un même outil une grande variété d'architectures d'agents et de modèles de comunication.

Nous prévoyons de continuer ce travail dans trois directions. D'une part, nous continuons à raffiner et étendre l'infrastructure de base formée par le noyau et les divers agents "systèmes", afin de mettre en place de nouveaux protocoles et en particulier de raffiner les mécanismes de meta-contrôle de la structure organisationnelle. D'autre part, nous cherchons à faire le lien entre notre travail sur les approches organisationnelles pour la méthodologie de systèmes multi-agents et des outils de constructions graphique comme SEdit. Finalement, nous continuons à évaluer cette plate-forme dans divers contextes applicatifs, et à mettre en place de nouveaux modèles d'agents.

A Disponibilité

La plate-forme MADKIT, l'outil SEDIT, ainsi que diverses extensions sont disponibles sur `<http ://www.madkit.org>`

Références

[1] Joachim Baumann and Nikolaos Radouniklis. Agent groups in mobile agent systems. In *IFIP WG 6.1, International Conference on Distributed Applications and Interoperable Systems (DAIS 97)*, 1997.

[2] Per Bothner. Functional scripting languages for the jvm. In *3rd Annual European Conference on Java and Object Orientation*, Arhus, Denmark, 1999.

[3] F. Bousquet, I. Bakam, H. Proton, and C. Le Page. Cormas : Common-pool resources and multi-agents systems. *Lecture Notes in Computer Science*, 1416 :826–837, 1998.

[4] M. H. Coen. SodaBot : A software agent environment and construction system. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, May 1994.

[5] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Third International Conference on Multi-Agent Systems (ICMAS '98) Proceedings*, pages 128–135. IEEE Computer Society, 1998.

[6] Jacques Ferber and Olivier Gutknecht. Operational semantics of multi-agent organizations. In *Workshop on Agent Theories, Architectures and Languages (ATAL'99)*. to be published in Intelligent Agents Series, Springer-Verlag, 1999.

[7] FIPA. Agent Management specification. 1997.

[8] Ernest J. Friedman-Hill. *Jess, The Java Expert System Shell*. Distributed Computing Systems, Sandia National Laboratories, Livermore, CA, 2000.

[9] H. Robert Frost. Java Agent Template. JAT Web page : http ://cdr.stanford.edu/ABE, 1997.

[10] General Magic Inc. Odyssey. Web page : http ://www.genmagic.com/agents/odyssey.html, 1997.

[11] Olivier Gutknecht and Jacques Ferber. Vers une méthodologie organisationnelle pour les systémes multi-agents. In Marie-Pierre Gleizes, editor, *Actes des Journèes Francophones en Intelligence Artificielle Distribuèe et Systémes Multi-Agents 1998*. Hermés, Nov 1999.

[12] Olivier Gutknecht, Jacques Ferber, and Emmanuel Lieurain. Des modèles hétérogènes de simulation par systèmes multi-agents. In Nils Ferrand, editor, *Actes du colloque Modèles et Systèmes multi-agents pour la gestion de l'environnement et du territoire (SMAGET) 1998*. Cemagref, Oct 1998.

[13] Bryan Horling and Victor Lesser. A reusable component architecture for agent construction. Master's thesis, University of Massachusetts/Amherst, 1998.

[14] Nelson Minar, Rogert Burkhart, Chris Langton, and Manor Askenazi. The Swarm simulation system : A toolkit for building multi-agent simulations. Technical Report 96-06-042, SFI, 1996.

[15] Hyacinth Nwana, Divine Ndumu, Lyndon Lee, and Jaron Collis. Zeus : A toolkit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1) :129–186, 1999.

[16] Jeremy Pitt and Abe Mamdani. Some remarks on the semantics of FIPA's agent communication language. *Autonomous AGents and Multi-Agent Systems*, 2(4), november 1999.

[17] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, and Richard Sanzi. Mach : A foundation for Open Systems. In *Proceedings of the 34th Computer Society Ithe Second Workshop on Workstation Operating Systems(WWOS2)*, September 1989.

[18] Mitchel Resnick. *Turtles, Termites, and Traffic Jams : Explorations in Massively Parallel Microworlds*. MIT Press, 1994.

[19] Juliette Rouchier, Olivier Barreteau, Frangois Bousquet, and Hubert Proton. Evolution and co-evolution of individuals and groups in environment. In *Third International Conference on Multi-Agent Systems (ICMAS '98) Proceedings*. IEEE Computer Society, 1998.

[20] Chris Zimmermann, editor. *Advances in object-oriented meta-level architectures and reflection*. CRC, 1996.