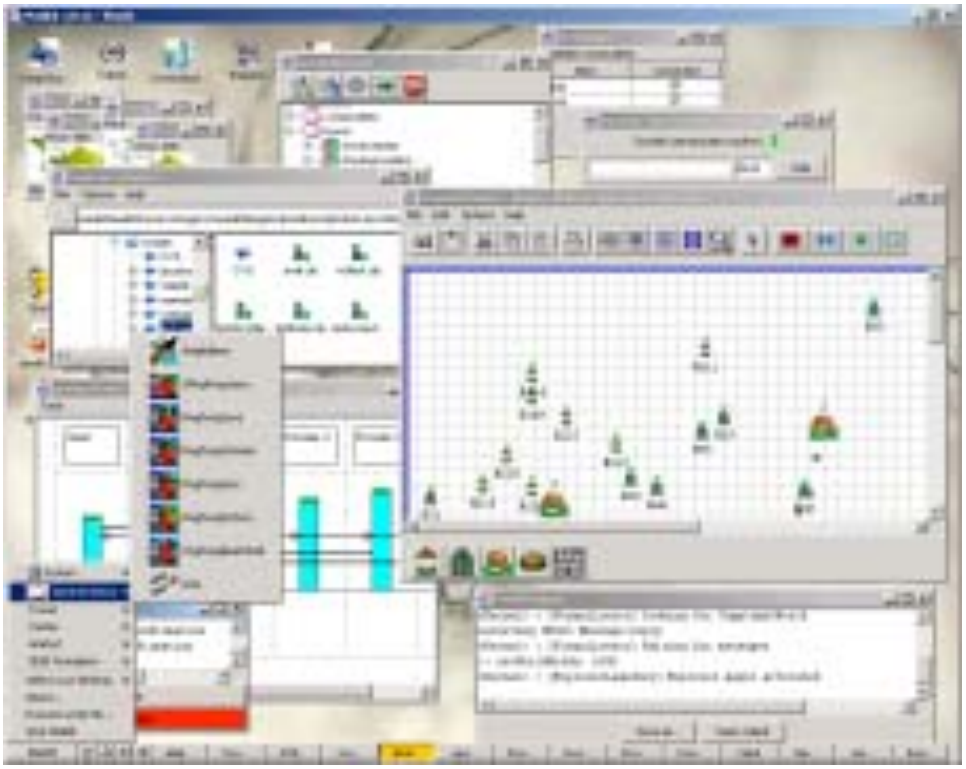
MadKit pas à pas

Démarrage et prise en main du logiciel MadKit



Jacques Ferber
LIRMM - Université de Montpellier II
ferber@lirmm.fr
v1.2 Avril 2009

MADKIT PAS À PAS

Démarrage et prise en main du logiciel MadKit

v1.1 - Mars 2009

I. Présentation générale

MadKit est une plate-forme de développement de systèmes multi-agents destinée au développement et à l'exécution de systèmes multi-agents et plus particulièrement à des systèmes multi-agents fondés sur des critères organisationnels (groupes et rôles).

MadKit n'impose aucune architecture particulière aux agents. Il est ainsi possible de développer aussi bien des applications avec des agents réactifs (comme le fait TurtleKit) que des agents cognitifs et communicationnels, et même de faire interagir aisément tous ces types d'agents. Cela permet ainsi aux développeurs d'implémenter l'architecture de leur choix.

MadKit est écrit en Java et fonctionne en mode distribué de manière transparente à partir d'une architecture "peer to peer" sans nécessiter de serveur dédié. Il est ainsi possible de faire communiquer des agents à distance sans avoir à se préoccuper des problèmes de communication qui sont gérés par la plate-forme.

Une application MadKit peut s'exécuter en mode distribué sans qu'il y ait besoin de modifier une seule ligne de code. Le mécanisme de distribution est indépendant de MadKit et il est possible d'en créer de nouveaux si les besoins s'en font sentir ou même d'utiliser des plate-formes distribuées existantes, telles que JXTA.

MadKit est construit autour du concept de "micro-noyau" et "d'agentification de services". Le micro-noyau de MadKit est très petit (moins de 100Ko de code), car il ne gère que les organisations (groupes et rôles) et les communications à l'intérieur de la plate-forme. Le mécanisme de distribution, les outils de développement et de surveillance des agents, les outils de visualisation et de présentation sont des outils supplémentaires qui viennent sous la forme d'agents que l'on peut ajouter au noyau de base.

De ce fait, MadKit peut être utilisé aussi bien comme outil de développement d'applications que comme un noyau d'exécution de système multi-agent qui peut être embarqué dans des applications quelconques.

MadKit est un logiciel libre de type "Open Source" avec une licence mixte GPL/LGPL. LGPL pour le micro-noyau et les outils de communication, et GPL pour les outils de développement. On peut ainsi facilement développer à l'aide de MadKit et ensuite utiliser les agents ainsi construits dans des applications commerciales.

On reconnait généralement à MadKit les qualités suivantes:

- Simplicité de mise en oeuvre et de prise en main,
- Intégration très facile de la distribution au sein d'un réseau.

- L'aspect pratique et efficace des concepts organisationnels pour créer différents types d'applications
- Hétérogénéité des applications et des types d'agents utilisables: on peut faire tourner sur MadKit aussi bien des applications utilisant des agents réactifs simples de type fourmis (plus de 50000 agents ayant un comportement simple ont tourné en MadKit), que des applications disposant d'agents cognitifs sophistiqués.

II. Installation et prise en main

Installation

Télécharger MadKit sur www.madkit.net (utilisez la version stable de préférence). Dézippez l'archive et placez MadKit où bon vous semble. MadKit est alors prêt à l'emploi.

ATTENTION: le designer de MadKit a besoin de se relier au compilateur Java. Il faut donc que vous ayez une installation du jdk de Java. Vérifiez que vous avez bien un

Sous **Linux** (la plupart des distributions), tapez dans le fichier ~/.bashrc les lignes suivantes:

```
JAVA_HOME=/chemin_ou_se_trouve_le jdk
export JAVA_HOME
PATH=$PATH:$JAVA_HOME/bin
export PATH
```

Sous **MacOS**: tout est normalement déjà installé, et la variable JAVA_HOME n'est pas utilisée.

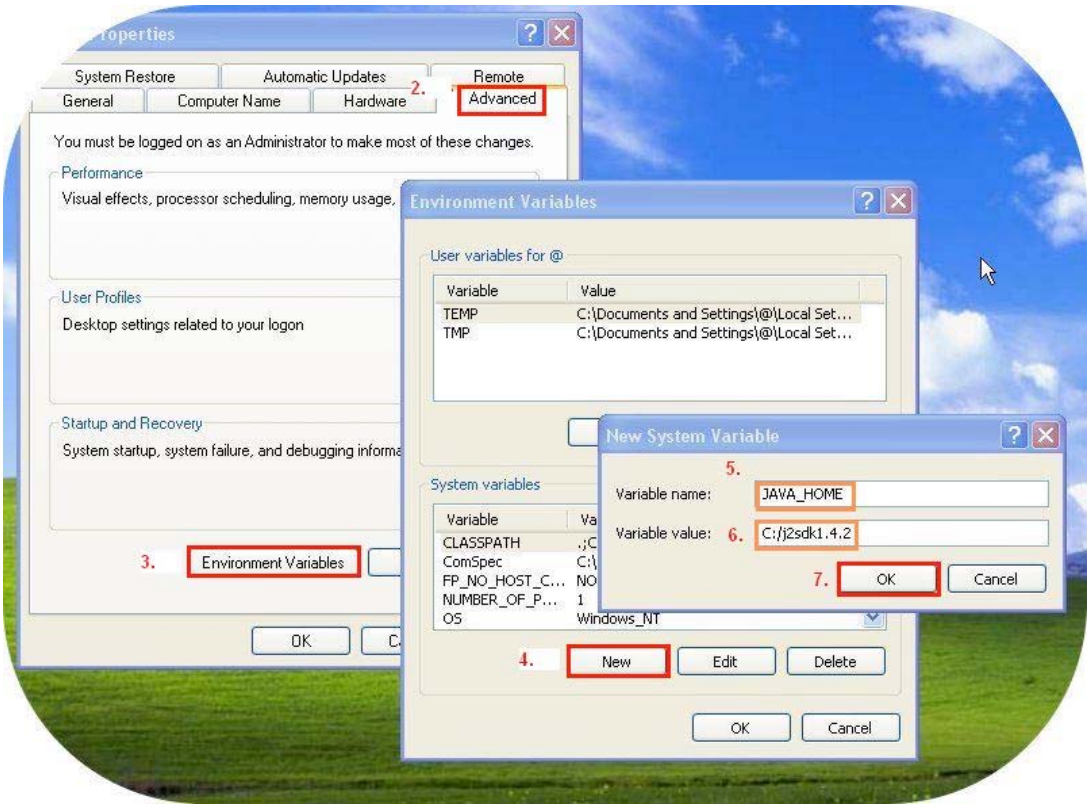
Sous **Windows**: déterminez le dossier d'installation du jdk (pas le JRE qui ne contient pas le compilateur, mais le JDK). Attention, il est facile d'avoir plusieurs jdk sous Windows, et c'est le dernier qui est utilisé par Windows.

Pour changer la variable JAVA_HOME sous Windows

1. Clic droit sur l'icône 'station de travail' et cliquer sur 'Propriétés'.
2. Cliquer sur l'onglet 'Avancé'.
3. Cliquer sur le boutn 'Variables d'environnement'.
4. Sous 'Variables Systèmes', cliquer 'New'.
5. Entrer le nom de la variable JAVA_HOME.

6. Mettre comme valeur de la variable le chemin du JDK et valider.
7. Cliquer sur 'Appliquer les changements'.

La figure suivante montre l'affectation de la variable JAVA_HOME (en anglais)



Attention: si lorsque vous tapez 'javac -version' une erreur apparaît c'est que vous devez aussi ajouter le chemin <chemin du jdk>/bin dans la variable d'environnement PATH...

Et finalement, si cela ne marche pas, vous pouvez copier le fichier *tools.jar* qui se trouve dans le JDK et placer la copie dans le dossier <madkit>/lib et le tour est joué. Mais c'est moins propre.

Démarrage

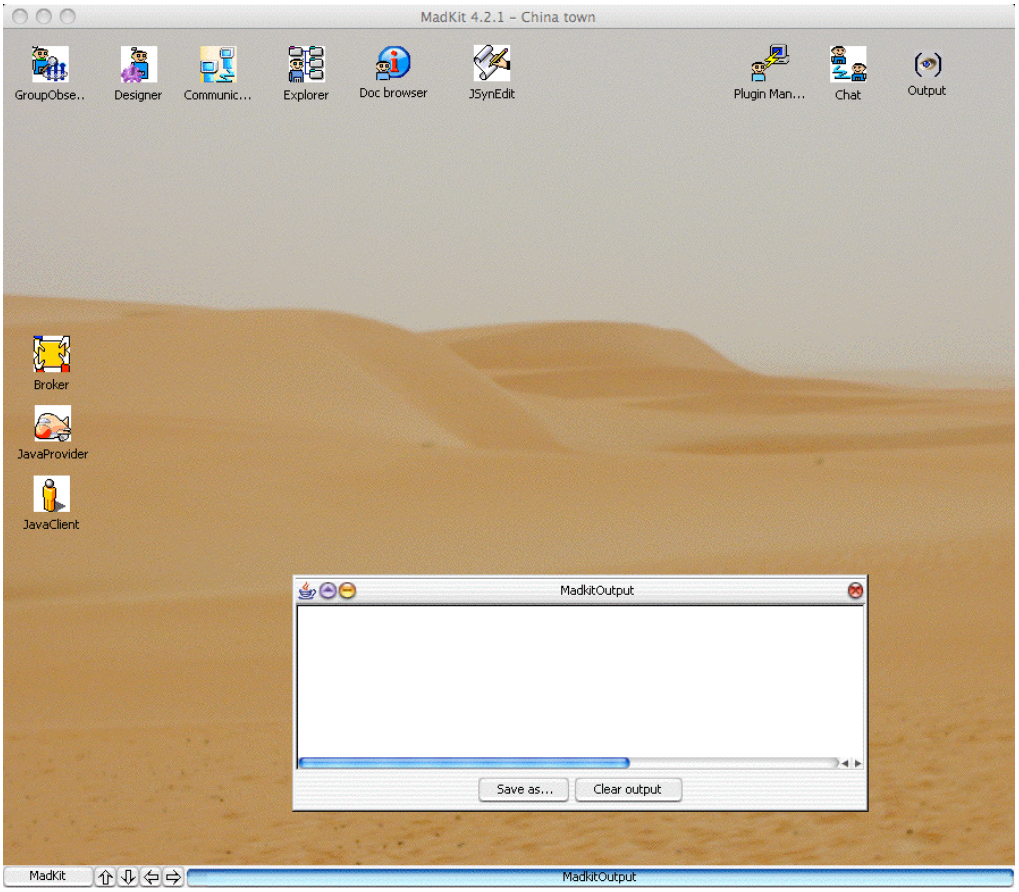
MadKit peut fonctionner de diverses manières, mais la solution la plus simple consiste à démarrer le desktop de MadKit qui permet de visualiser facilement les agents et qui comprend de nombreux outils de développement et de surveillance d'agents.

Allez dans le dossier `bin` de MadKit.

- Sous Windows: double cliquez sur `madkit.exe`
- Sous Mac OS: double cliquez sur `madkit.app`

- Sous Linux, Mac OS ou tout logiciel sous Unix: tapez ./desktop.sh dans une fenêtre de console (ou terminal).

Vous verrez apparaître alors la fenêtre suivante qui correspond à ce qu'on appelle le "Desktop":



Le Desktop est l'environnement de travail standard sous MadKit. On peut en modifier son apparence grâce à des "skins" différents, on peut ajouter et supprimer des icônes à loisir pour l'adapter à ses besoins.

C'est à partir du Desktop que l'on peut créer, lancer et arrêter des agents, ainsi que surveiller leur fonctionnement.

Les agents principaux

Il existe un certain nombre d'outils qui sont disponibles sous MadKit et qu'il est nécessaire de connaître pour bien travailler. Ces outils sont les suivants

1. Designer: permet de concevoir de nouveaux projets et facilite la création d'agents grâce à des templates pré-définis. C'est le point d'entrée pour la création de nouveaux projets et de nouveaux agents sous MadKit. C'est aussi le point de départ pour pouvoir lancer facilement des agents.

2. GroupObserver (et le Spy): permet d'observer la structure organisationnelle de MadKit, c'est-à-dire l'ensemble des groupes et des rôles. Permet aussi de tracer les messages qui ont lieu entre les agents et notamment tous les messages à l'intérieur d'une communauté ou d'un groupe.

3. Explorer : correspond à un simple explorateur de fichiers. Permet de lancer et d'éditer des agents à partir de là. Depuis l'arrivée du Designer, on utilise en général moins l'explorateur, mais il peut continuer à répondre à des besoins ponctuels.

4. DocBrowser: c'est là que se trouve toute la doc de MadKit. Indispensable pour connaître les primitives de base. Pour programmer de nouveaux agents, il est utile d'aller soit dans la doc du développeur (Developper's guide) et dans les API, notamment 'madkitkernel>AbstractAgent' et 'madkitkernel>Agent' qui contiennent la plupart des primitives importantes pour le développement d'applications génériques sous MadKit. Pour développer dans des applications ciblées (TurtleKit, Warbot, etc..) il est préférable de s'adresser à la documentation liée à ces applications.

5. JSynEdit: c'est un éditeur générique avec colorisation syntaxique, qui est tiré de Jext et de l'ancienne version de JEdit, des éditeurs en OpenSource. Ces éditeurs ont été adaptés à MadKit pour servir d'outil d'édition rapide.

6. Communicator: c'est l'outil qui permet de faire communiquer les agents entre eux. Cet outil est simple et fiable, très adapté à des applications intranet (il ne passe pas les firewall ni les proxys) allant de 2 à une petite vingtaine de noyaux interconnectés. Au delà, les algorithmes de synchronisation des informations entre noyaux MadKit ne sont plus adaptées.

7. Editor: pratique pour regarder rapidement un bout de code, ou pour lancer des messages de type "chaînes de caractères" (`StringMessage`) à des agents.

8. Les évaluateurs de langages de commande: permet de taper des commandes dans un langage de script (Python, BeanShell, Scheme, etc..) et de lancer des agents de manière interactive directe. Très pratique pour scripter des sessions.

9. La fenêtre Output qui affiche tous les messages de sortie de MadKit. Cette fenêtre vous indique en particulier lorsqu'il y a des problèmes lors de la compilation ou de l'exécution d'agents.

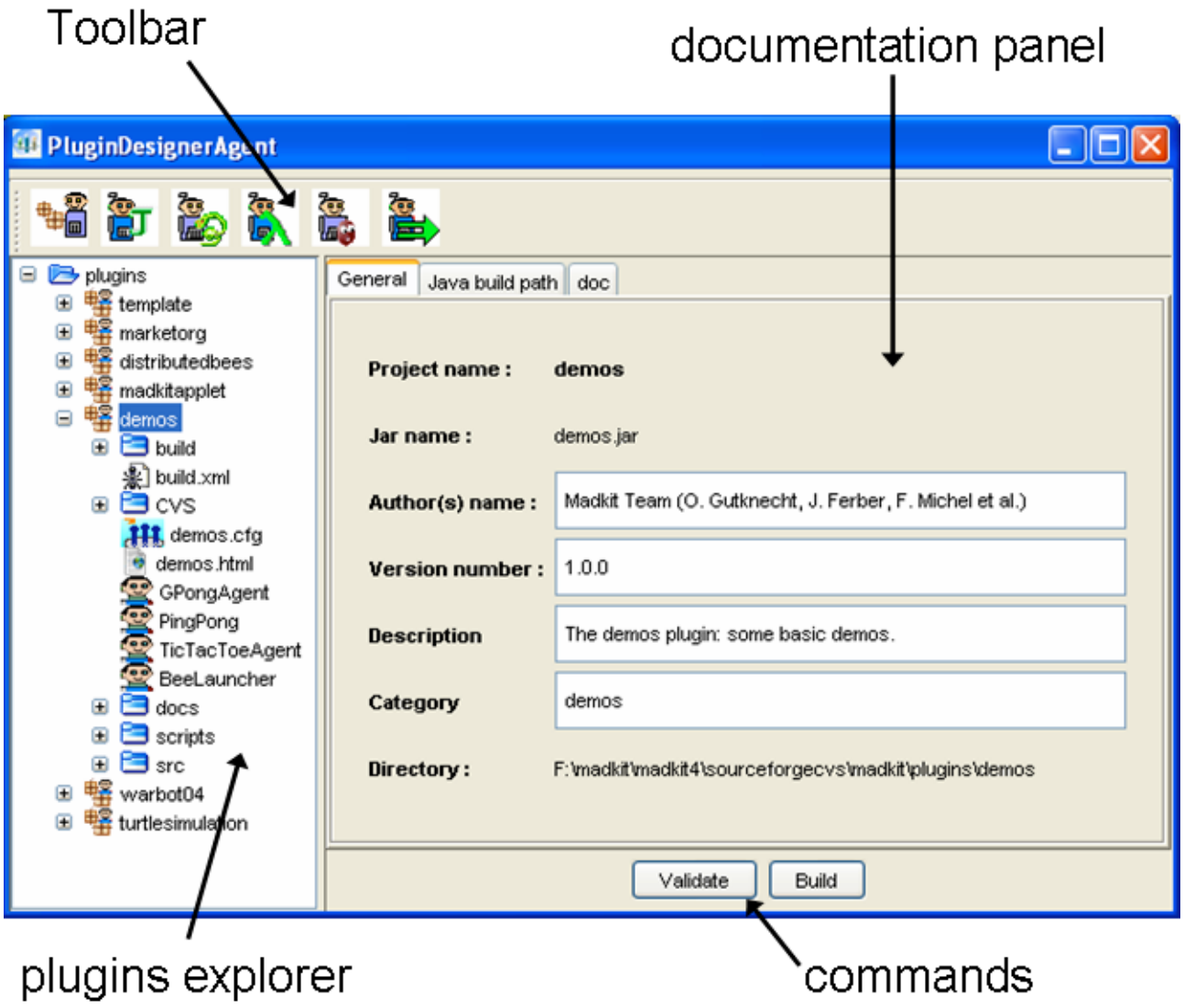
Note: tous ces outils sont en fait des agents MadKit disposant de certaines capacités. De ce fait, tous ces outils apparaissent dans le GroupObserver dans le groupe `System` avec leurs rôles respectif.

III. Créer son premier projet

Pour créer un projet, il faut cliquer sur l'icône Designer qui se trouve sur le Desktop ou sélectionner l'icône du Designer dans le menu MadKit.

Cela lance un agent qui montre tous les plugins qui peuvent être modifiés et mis à jour (les plugins "système" qui font partie du noyau ne sont pas visualisés et il n'est donc pas possible de les modifier à partir du Designer).

Il y a quatre parties dans l'interface du Designer:



- Le "Plugin explorer" qui comprend une arborescence de l'ensemble des plugins disponibles ainsi que les composants de chacun de ces plugins.

- Le panneau de documentation qui affiche les propriétés du plugin.
- Le panneau de commande, situé en dessous, qui comprend deux boutons: `Validate`, qui valide les modifications apportées dans les paramètres du plugin et `Build` qui sert à régénérer un projet (compilation, création de jar, chargement des nouvelles classes, etc..).
- La barre d'outils, située au dessus de l'arborescence des plugins, et qui comprend les opérations de création de base: création d'un nouveau plugin et création d'agents.

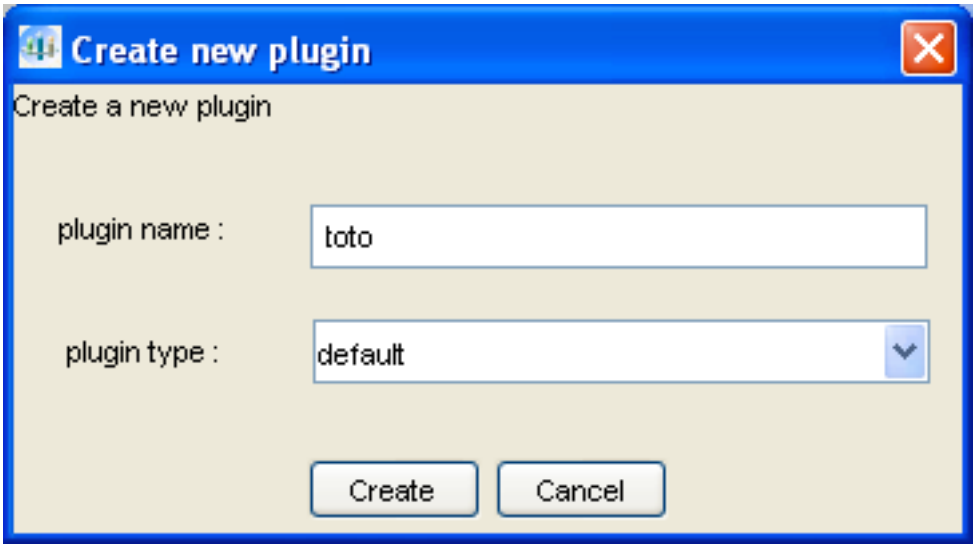
Pour sélectionner un plugin, il suffit de double cliquer sur son icône dans l'arborescence des plugins. Cela montre à la fois ses composants mais aussi cela affiche ses propriétés dans le panneau de documentation.

Créer un nouveau projet

Tous les projets sous MadKit prennent la forme de plugins. De ce fait le terme "projet" ou "plugin" peuvent être utilisés indifféremment.

Pour créer un nouveau plugin, il suffit de cliquer sur l'icône qui se situe dans la barre d'outil.

Cela ouvre le dialogue suivant:



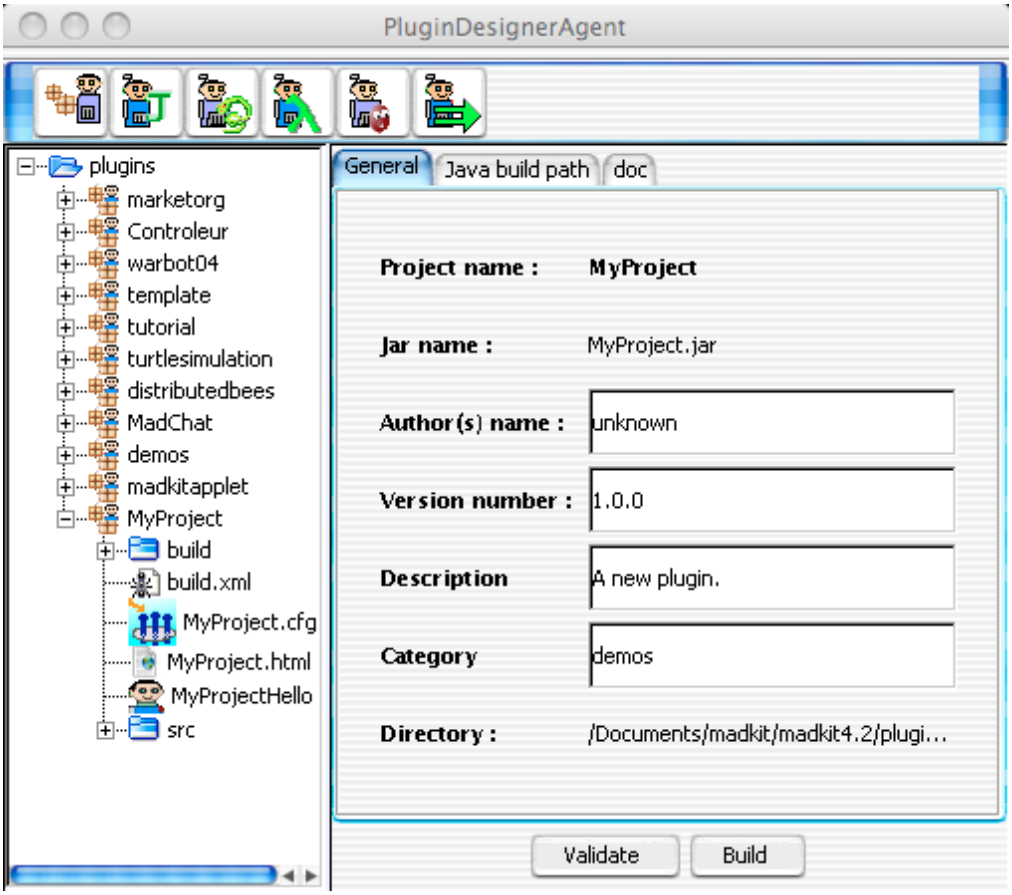
Le premier champ correspond au nom du projet (ou plugin). Le second champ est une liste déroulante qui indique les types de plugins qui peuvent être réalisés à l'aide du designer.

Les projets possibles actuellement sont de l'ordre de trois: default (projets standards), TurtleKit (projets qui fonctionnent sous TurtleKit) et Warbot (projet permettant de réaliser des agents sous Warbot). D'autres seront disponibles avec le temps.

Note: attention, il n'est pas possible de démarrer un projet Warbot directement, à partir du Designer. Adressez vous auprès de la documentation de Warbot pour créer un projet Warbot.

Nous supposerons ici que vous créez un projet de type 'Default' qui correspond à des agents standards sous MadKit.

Par exemple, si l'on crée le projet 'MyProject' on voit apparaître l'arborescence suivante:



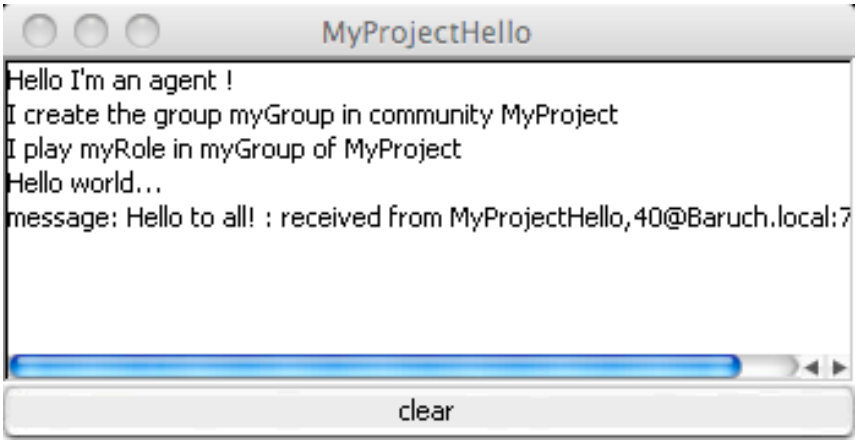
Note: vous verrez apparaître dans la fenêtre "Output" le message "Build successful" qui signifie que d'une part le plugin a bien été créé, mais aussi que les agents de démonstration ont bien été créés et compilés.

Le plugin comprend le contenu suivant:

- Les dossiers `src` et `build` qui contiennent respectivement les sources (.java files) and les .class du plugin.
- Le fichier `build.xml` qui est un fichier 'ant' qui est utilisé pour compiler le projet, générer le .jar, etc.. (Ant est une sorte de 'makefile' écrit en Java pour des programmes Java. Il est très utilisé dans les outils de développement Java (IDE) tels qu'Eclipse, pour créer et mettre à jour les projets. Voir le site de 'ant' ant.apache.org pour avoir plus d'information à ce sujet).

- le fichier `MyProject.cfg` qui permet de lancer tout un ensemble d'agents en même temps (pour plus d'information, lire la documentation dans le "user's guide" et le "developper's guide" à ce sujet).
- le fichier `MyProject.html` qui est une page générée et qui contient une applet liée au plugin (pour plus d'information, lire la documentation dans le "user's guide" et le "developper's guide" à ce sujet).

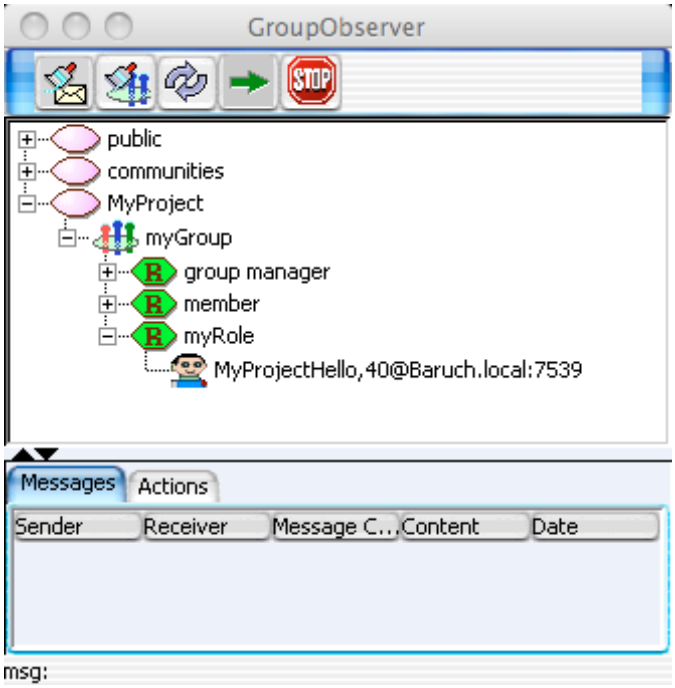
Si vous cliquez sur l'icône de l'agent `MyProjectHello`, vous lancerez un agent MadKit de base, qui permet de démarrer dans votre projet.



Lors de son fonctionnement, l'agent indique qu'il est entré dans la communauté MyProject (la communauté associée au plugin), mais aussi dans le groupe "myGroup" avec le rôle "myRole".

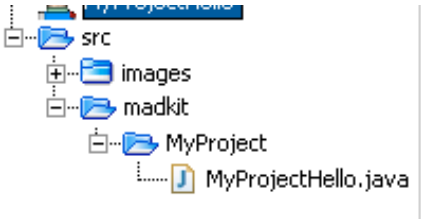
Il est possible de voir son fonctionnement en utilisant le GroupObserver. Double cliquez l'icône du GroupObserver:

Cela ouvre l'agent qui permet d'observer la participation des agents aux groupes et aux rôles, comme le montre la figure ci-dessous:



On peut constater que l'agent `MyProjectHello` qui se trouve sur l'ordinateur `Baruch.local` (celui de l'auteur de ces lignes au moment où elles ont été écrites) se trouve bien dans le groupe `myGroup` de la communauté `MyProject`, sous le rôle `myRole`.

Toutes les actions des agents se trouve dans le source de l'agent. Dans le designer allez dans le dossier `src` de votre projet et ouvrez les dossier jusqu'à voir apparaître le fichier `MyProjectHello.java`.



Cliquez sur ce fichier avec un click droit de souris et sélectionnez 'edit with JSynEdit'. Cela ouvre l'éditeur JSynEdit qui comprend un système de colorisation syntaxique et qui permet d'éditer facilement du code dans l'un des langages de MadKit et notamment Java.

Vous pouvez alors modifier le code de votre agent, Par exemple, si vous voulez qu'il affiche le code `"Bonjour à tous"` en lieu et place du message "`Hello World`", allez dans le code de l'agent et modifiez le code ainsi:

```
public void live()
 {
      println("Bonjour à tous"); // code modifié
      broadcastMessage(myCommunity, myGroup, myRole,
                           new StringMessage("Hello to all!"));
```

```
        while(alive){
                Message m = waitNextMessage();
                handleMessage(m);
        }
}
```

Il suffit ensuite de sauvegarder le fichier puis de mettre à jour le projet en cliquant sur le bouton 'Build', ce qui compile les fichiers, crée le jar et recharge les nouvelles classes.

En cliquant sur l'icône de MyProjectHello c'est le nouveau comportement qui sera exécuté.

Note: les anciens agents continueront à exécuter les classes anciennes, c'est-à-dire que seuls les agents créés après le Build bénéficieront des modifications de code.

Ajouter de nouveaux agents

Pour créer de nouveaux types d'agents en utilisant Designer, il suffit de cliquer sur l'un des autres boutons de sa toolbar.

Agent Java

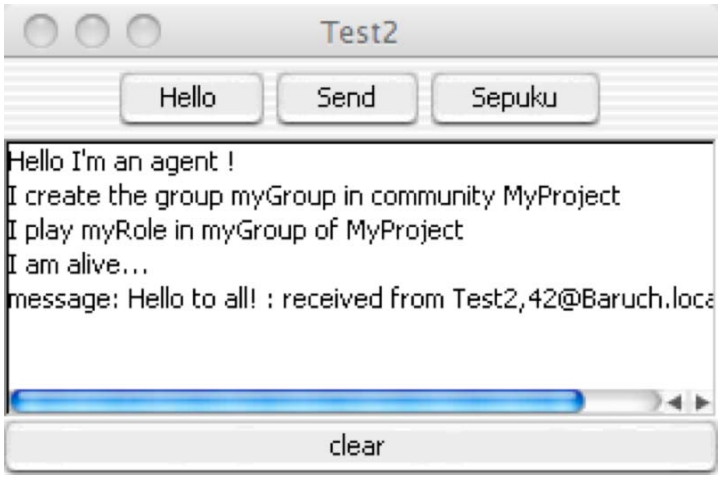
Création d'un agent Java

Ce bouton permet de créer un nouvel agent Java qui sera compilé lors d'un Build.

Il existe plusieurs templates d'agents Java. Pour l'instant deux templates sont disponibles, mais d'autres seront fournis dans le futur. Les templates existant sont:

1. Standard Java agent
2. Standard Java agent with GUI

Le premier correspond à la structure de l'agent 'Hello' qui est créé à l'initialisation du projet. Le second offre une interface graphique à l'agent. Cette interface graphique comprend trois boutons et un champ d'affichage. Les boutons servent respectivement à afficher Hello dans le champ d'affichage, à envoyer des messages "Hello" à tous les agents jouant le même rôle dans le même groupe et la même communauté, et le troisième, appelé 'Sepuku' à tuer l'agent qui s'envoie à lui-même un message pour se suicider.



En fait deux classes sont créées: la classe de l'agent lui-même, ici Test2 dans l'exemple, et la classe de l'interface graphique, ici Test2GUI.

Il est bien entendu possible (et même souhaitable) de modifier cet agent et son interface pour l'adapter à vos besoins.

Agents Scripts

Création d'un agent "script"

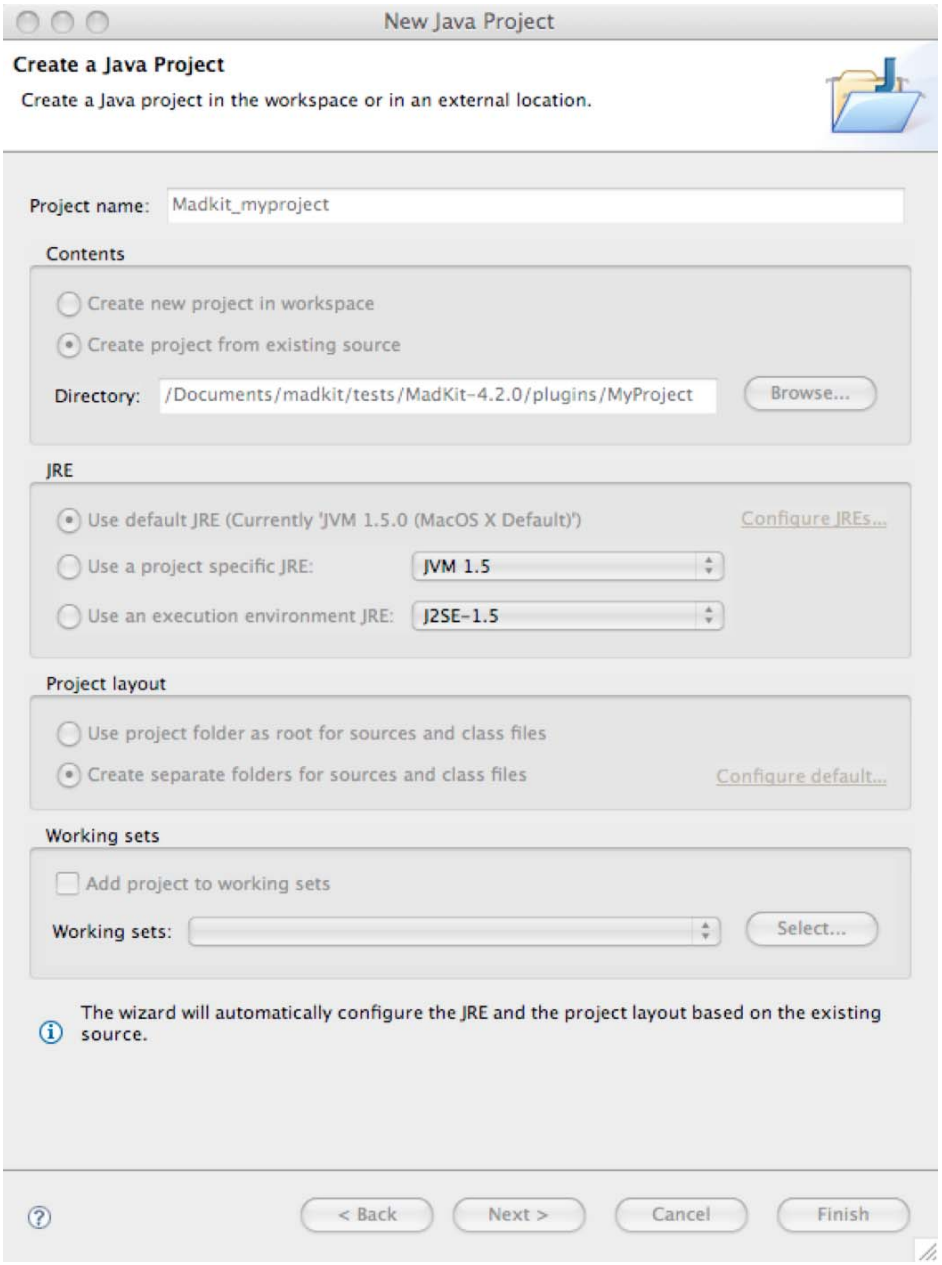
Ces boutons servent à créer d'agents "scripts" c'est-à-dire des agents dont le code est interprété par l'un des langages de script disponible sous MadKit, à savoir: Scheme, BeanShell, Jess et Python. Ces agents sont créés dans le dossier 'script' du projet et n'ont pas besoin d'être compilés.

Note: Sous Designer, il peut être nécessaire de faire un 'Refresh' du projet pour voir ces fichiers apparaître dans le dossier 'scripts'. Il suffit de faire un clic droit de la souris sur le noeud du plugin dans l'arborescence et de choisir, dans le popup menu qui apparaît, l'item 'Refresh'.

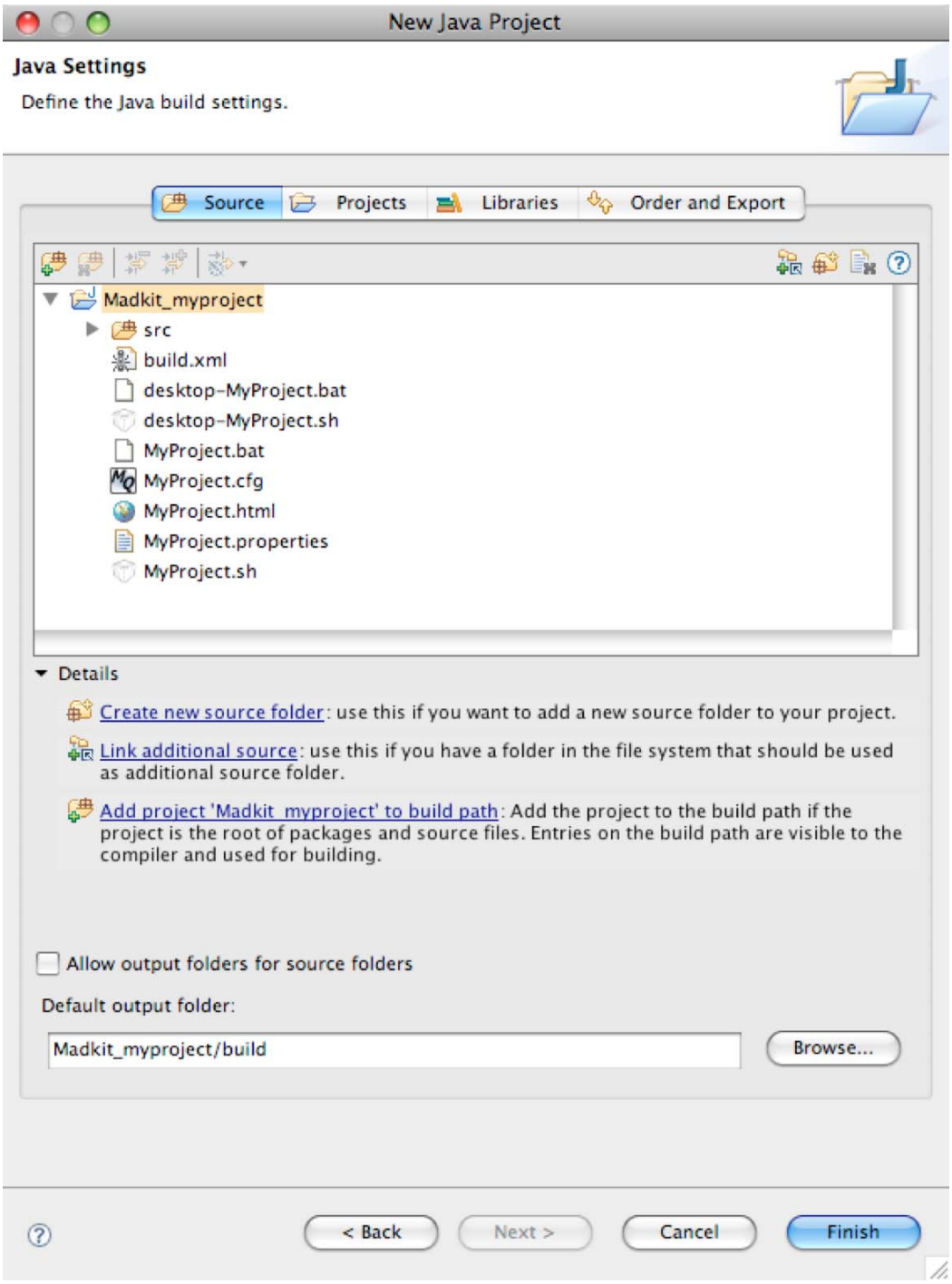
Utiliser Eclipse pour compiler un plugin de MadKit

Il est très facile d'utiliser Eclipse pour créer un projet sous MadKit. Il suffit que le projet Java associé à Eclipse pointe vers le plugin créé sous le PluginDesigner de MadKit et d'inclure dans le classpath du compilateur d'Eclipse les jars dont on a besoin.

Voici comment, pas à pas, il est possible de créer un projet Eclipse qui pointe vers un plugin de MAdKit. Supposons que nous ayons créé le projet 'MyProject' comme on l'a vu ci-dessus. En créant 'new Java project' depuis Eclipse, on obtient le dialogue suivant:

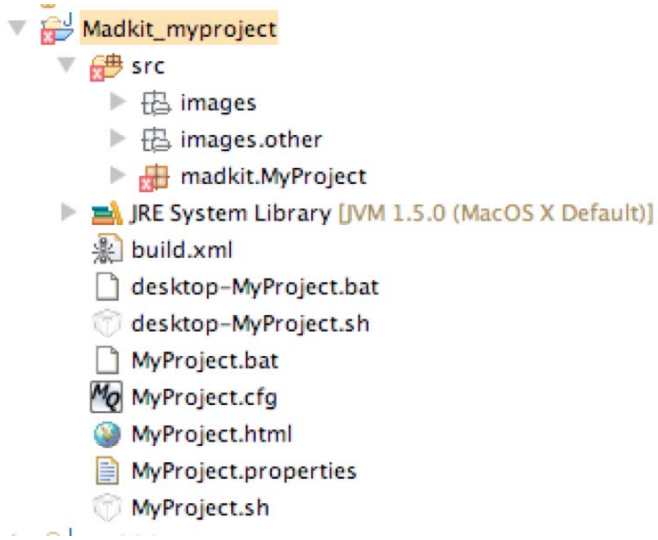


Il faut alors cliquer sur 'create project from existing source' et entrer le chemin d'accès au plugin. Puis, on clique sur 'next' ce qui ouvre la page de dialogue suivante où l'on peut constater que le projet est bien pris en compte par Eclipse:

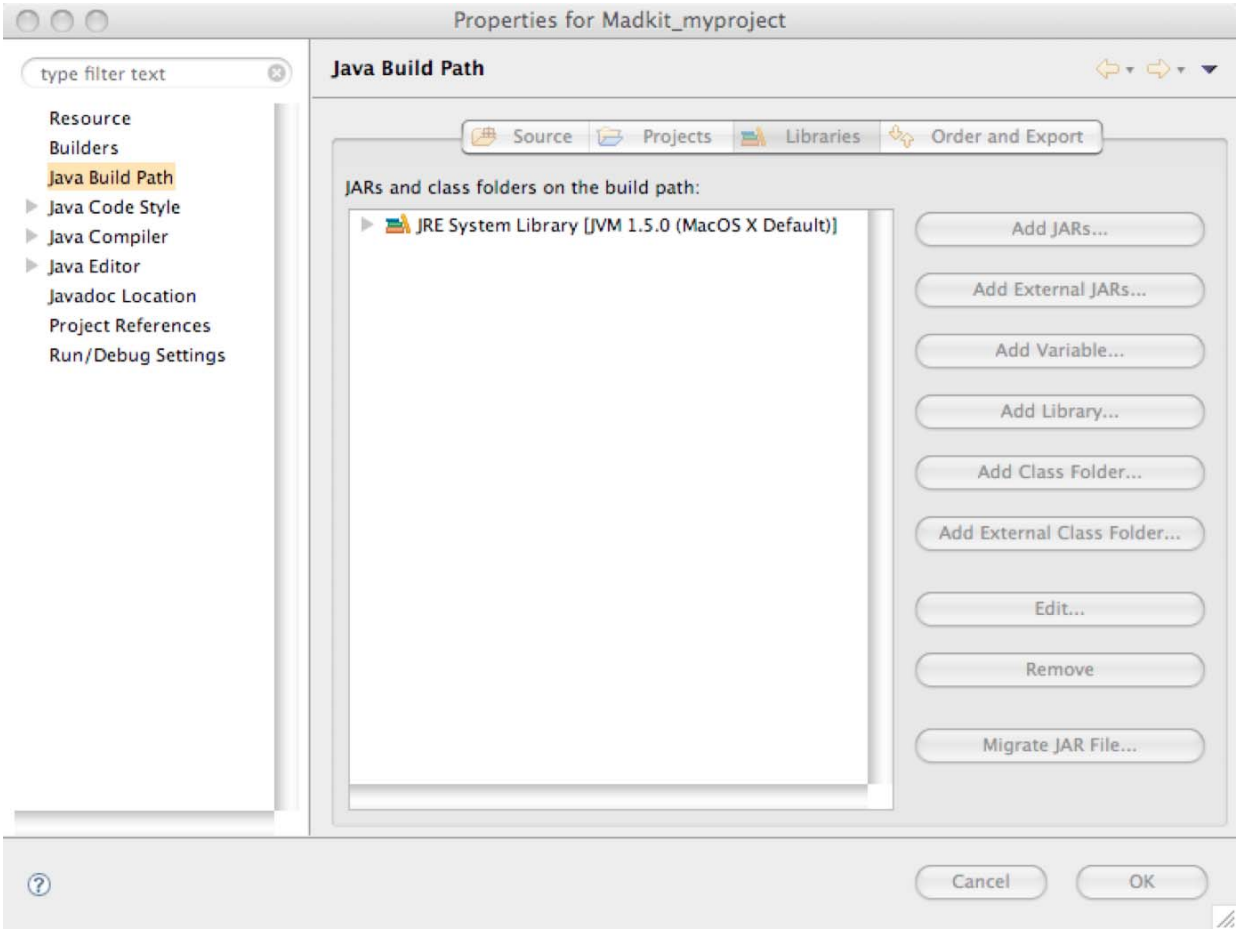


A partir de là, il suffit de cliquer sur 'Finish' pour que Eclipse crée un projet qui pointe bien vers votre plugin.

Si l'on regarde dans le 'package explorer' de Eclipse, on peut constater que le projet comprend des erreurs, comme le montre la figure suivante:



Cela est dû au fait que le classpath de Eclipse ne contient pas les bons jars dont il a besoin pour compiler le projet. Pour ajouter les jars nécessaires, il suffit de faire un click droit de souris, de sélectionner 'properties' et d'ouvrir le dialogue des propriétés du projet. Cliquez ensuite sur 'java build path' et 'libraries'. Vous obtenez l'écran suivant:

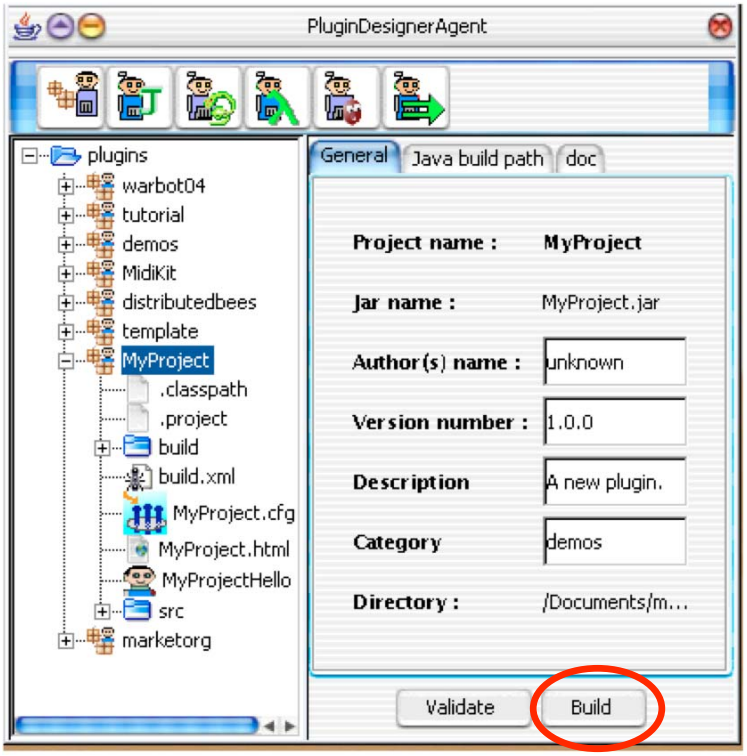


Cliquez alors sur 'add external jar' et ajouter les jars nécessaires dans le dossier 'madkit/lib'. Votre projet a au moins besoin de 'madkitkernel.jar' (sans cela rien ne fonctionne), mais d'une manière générale les projets par défaut ont besoin de 'madkitutils.jar' et de 'messages.jar'.

Vous pouvez rajouter des jars, mais ceux ci doivent être aussi présents dans le `build.xml` de votre plugin MadKit, comme cela est indiqué dans la documentation du PluginDesigner (double cliquez sur DocBrowser et sélectionnez plugins>designer).

Pour mettre à jour votre projet MadKit sous Eclipse

Eclipse se charge de la compilation des fichiers sources. Mais pour mettre à jour votre projet vous devez cliquer sur le bouton 'Build' correspondant à votre projet dans le PluginDesigner:



IV. AGR: la gestion des organisations sous MadKit

MadKit considère que les agents sont des entités qui vivent au sein d'organisation. Le modèle AGR, que nous avons présenté, Olivier Gutknecht et moi, en 1998, repose sur un ensemble minimal de concepts: les concepts d'agent, de groupes, et de rôles. Dis ans après ces concepts sont toujours aussi vivants, et ils ont montré leur importance dans le domaine du génie logiciel orienté agents. AGR est ainsi devenu une référence incontournable dans le domaine des systèmes multi-agents organisationnels (ou OMAS pour Orgaizational Multi-Agent Systems). De-

puis, d'autres modèles ont été proposés, mais AGR reste encore la base pour la compréhension des organisations dans des systèmes multi-agents.

Introduction à AGR

La particularité d'AGR c'est de ne rien supposer quant à l'architecture ou au mode de fonctionnement d'un agent. Voici les principales définitions d'AGR

- Un **agent** est une entité informatique qui envoie et reçoit des messages, qui peut entrer (et sortir) de groupes et qui joue des rôles dans ces groupes. Cette entité est autonome et communicante.
- Un **groupe** est constitué de rôle et regroupe des agents qui ont soit un aspect en commun (par ex. des agents qui possèdent le même langage de communication), soit un ensemble d'agents qui travaillent ensemble, qui collaborent à une même activité (par ex. des agents qui essayent de résoudre un problème).
- Un **rôle** correspond soit à un statut d'un agent, soit à une fonction particulière qu'un agent joue dans un groupe. Les rôles sont locaux aux groupes et n'ont de sens qu'à l'intérieur d'un groupe. Un rôle ne peut pas être à cheval sur deux groupes.

Trois concepts importants ont été à la base de tous les développements sous MadKit et doivent être considérés comme des guides de pensées (guidelines) dans tout développement concernant AGR.

1. **Communications organisationnelles**: Un agent A ne peut communiquer à un agent B que si A et B sont membres d'un même groupe G. De ce fait, on peut dire que A communique à B dans G. Malheureusement, cette contrainte n'a pas été implémentée directement à la base de MadKit, et il est effectivement possible (jusqu'aux versions 4.xx) pour un agent A de communiquer à un agent B même s'il n'existe pas de groupe commun. Mais de nouvelles versions de MadKit (à partir de la version 5.0) seront plus contraignantes à ce sujet, et il ne sera pas possible de communiquer à un agent si l'on ne vit pas dans le même groupe.
2. **Autonomie des agents**. Toutes les actions et comportements d'un agent viennent de l'agent lui-même. Il n'est pas possible de modifier l'état d'un agent ou de manipuler un agent depuis l'extérieur d'un agent. Si l'on veut que cet agent fasse quelque chose, il faut lui demander. Cela est particulièrement vrai pour tout l'aspect organisationnel de MadKit: un agent doit demander expressément et personnellement d'entrer dans un groupe ou de jouer un rôle, cela ne peut pas être fait à la place de l'agent ou depuis l'extérieur de l'agent. D'autre part, il est formellement interdit de pouvoir "entrer" dans un agent, de connaître directement ses états "mentaux" depuis l'extérieur de l'agent (sauf évidemment pour des raisons de mise au point ou de visualisation d'information, mais c'est le seul cas autorisé).

3. **Localisation organisationnelle**. Tout agent est situé dans l'espace organisationnel. Cela signifie que tout agent joue au moins un rôle dans un groupe. Bien que cet aspect n'ait pas été renforcé dans le noyau de MadKit (il est possible de créer des agents sans les associer à des groupes ou à des rôles), tous les exemples et programmes proposés dans MadKit observent cet impératif.

4. **Les aspects multiples d'un agent**. Un agent peut jouer plusieurs rôle et être membres de plusieurs groupes. Il peut ainsi jouer plusieurs rôles au sein d'un même groupe ou jouer plusieurs rôles dans des groupes différents. Tout cela est possible et il n'existe aucune contrainte à ce sujet. En suivant les principes d'autonomie de l'agent, c'est à l'agent de vérifier que les rôles qu'il jouent sont bien cohérents.

5. **L'opacité cognitive des agents**. Dans AGR comme nous l'avons noté plus haut, rien n'est imposé quant à la manière pour un agent de "penser" ou de prendre des décisions. Cela signifie qu'un agent réactif de type "fourmi" peut très bien communiquer avec un agent très complexe et très cognitif.

Création de groupes

Pour créer un groupe, il suffit, pour un agent, de le demander en utilisant la primitive createGroup :

```
int createGroup(
           boolean dist,      // indique si le groupe est distribuable
           String communauté, // nom de la communauté
           String groupe,     // nom du groupe ainsi créé
           String description, // description du groupe (éventuellement)
           Object vérificateur) // object vérificateur des entrées
```

`dist` indique si le groupe peut être réparti ou non sur le réseau. Si le `boolean` est à `true`, et si le Communicator est en fonctionnement (ou l'un des autres outils de distribution sous MadKit), alors le groupe sera automatiquement disponible sur toutes les plate-formes MadKit connectées.

La chaîne description n'est pas utilisée pour l'instant.

L'objet vérificateur permet de restreindre les entrées des agents aux seuls agents disposant d'une caractéristique, le test de cette caractéristique étant placée dans le vérificateur. Nous verrons l'utilisation de cette fonctionnalité plus loin dans le texte {{check}}.

Cette primitive retourne la valeur `1` si le groupe a pu être créé et `-1` si le groupe existait déjà.

Note: un groupe ne peut pas être détruit par un agent. Un groupe disparaît lorsque tous ses membres l'ont quitté (soit en le quittant explicitement par l'utilisation des primitives `leaveGroup` et `leaveRole`, soit en "mourant").

Entrer dans un groupe

Comme cela a été dit plus haut, les agents doivent demander de jouer un rôle dans un groupe. On entre toujours dans un groupe pour jouer un rôle, ce qui implique qu'on ne peut pas simplement "entrer dans un groupe", il faut toujours demander explicitement un rôle. Par défaut tous les membres d'un groupe ont aussi le rôle "member"

```
public int requestRole(String communityName,
                       String groupName,
                       String roleName,
                       Object memberCard)
```

retourne les valeurs suivantes:

`1` : operation success; `-1` : access denied; `-2` : the role is already handled by this agent; `-3` : the group does not exist; `-4` : the community does not exist.

La 'memberCar' est un objet qui peut être présenté pour entrer dans un groupe, si le groupe a été créé avec un objet vérificateur.

Communications sous MadKit

V. Ecrire ses agents sous MadKit

La création d'agents sous MadKit nécessite de bien comprendre ce que l'on cherche à produire et à bien identifier le type d'agents que l'on veut réaliser. Nous examinerons les points suivants:

6. Communications point à point et "broadcast"
7. Agents actifs vs agents passifs: comment créer des agents qui ont une activité indépendante de la réception des messages.
8. Requêtes et tâches: comment demander à un agent de faire quelque chose.

Communications point à point et broadcast

Agents passifs et agents actifs

Les agents passifs sont des agents qui n'agissent que lorsqu'ils reçoivent un message mais qui n'ont aucune activité particulière d'eux-mêmes.