

Rapport 2003 du projet

MODÈLE, AGENT, ASPECT ET COMPOSANT

Olivier Boissier Noury Bouraqadi Jean-Claude Royer Djamel Seriai
Christelle Urtado Sylvain Vauttier Laurent Vercouter

16 février 2004

1 Introduction

L'objectif du projet MODÈLE, AGENT, ASPECT ET COMPOSANT (MAAC) est de définir et d'implémenter un modèle de composant intégrant des caractéristiques de la programmation par aspect pour le développement d'infrastructures logicielles supports à des applications multi-agents. Ce projet contribue à la mise en place d'une première activité concrète de recherche et de valorisation coordonnée dans le domaine informatique entre les Ecoles des Mines d'Alès, Douai, Nantes et Saint Etienne. Les équipes et membres permanents de ce projet sont :

- *EM Alès : MOO* : Christelle Urtado et Sylvain Vauttier
- *EM Douai : GIP* : Djamel Seriai et Noury Bouraqadi
- *EM Nantes : OBASCO* : Pierre Cointe, Thomas Ledoux et Jean-Claude Royer
- *EM Saint Etienne : SIMMO/SMA* : Olivier Boissier et Laurent Vercouter

Ce document donne un aperçu des activités de chaque équipe, des points de convergence et de l'activité commune qui s'est mise en place depuis le début du projet. Celle-ci s'insère dans le domaine du logiciel (intelligence artificielle ou génie logiciel). Les thèmes abordés sont : *agent*, *aspect* et *composant*. Nous les détaillerons dans la suite en montrant les multiples interactions qui existent entre eux. Un planning et les perspectives sont présentées à la fin du document.

Ce projet a été initié au printemps 2003. Il a débuté par l'exploration, par chacun des partenaires, des directions de recherche définies dans le texte du projet. Une mise en commun des travaux a commencé à l'automne 2003 afin de constituer une première synthèse des résultats.

2 Contexte du projet

L'objectif du projet MODÈLE, AGENT, ASPECT ET COMPOSANT (MAAC) est de définir et d'implémenter un modèle de composant intégrant des caractéristiques de la programmation par aspects pour le développement d'infrastructures logicielles supports à des applications multi-agents. Ce modèle permettra aux développeurs d'applications une approche de haut-niveau, plus productive et plus sûre pour la conception d'une très grande variété d'applications, distribuées ou non. Le modèle sera implémenté en Java et intégré dans la plate-forme open source Eclipse développée par OTI/IBM. Ce projet est une suite scientifique du projet CMAS, entre Saint-Etienne et Nantes en 2002, abondé par l'ANVAR. Les partenaires de ce nouveau projet sont au nombre de quatre. Il s'agit des Ecoles des Mines d'Alès, Douai, Nantes et Saint-Etienne.

Dans ce qui suit nous rappelons brièvement ce que sont les agents, aspects et composants ainsi que quelques applications démonstratives de leurs intérêts.

2.1 Les systèmes multi-agents

Apparu dans les années 90, le domaine des Systèmes Multi-Agents est en plein essor actuellement. Evolution naturelle des paradigmes de programmation informatiques, il fusionne différentes technologies et se situe à la croisée de problèmes actuels tels que distribution, décentralisation, hétérogénéité, adaptation aux évolutions de l'environnement,... S'appuyant sur la *Programmation par Objets* et l'*Intelligence Artificielle*, il transforme chacun des objets en *agents autonomes*, par la capacité à décider localement des programmes et données à exécuter en réponses à des buts locaux et à des stimuli extérieurs. Il se situe également dans l'*Informatique Distribuée*. Cependant, privilégiant la décentralisation et la distribution des applications, il accentue la confrontation entre exécutions locales des agents et le fonctionnement global du système. Ce paradigme étend la notion de *composants logiciels* via l'introduction d'un couplage explicite entre agents au niveau de la connaissance au lieu d'être seulement au niveau des types de données.

Les SMAs proposent un ensemble de modèles et de concepts qui permettent aux concepteurs et utilisateurs d'applications multi-agents d'aborder la complexité des applications à la fois au sein de modules appelés *agent*, et au niveau du système lui-même, appelé *système multi-agent*. Ces deux concepts définissent deux niveaux de modélisation. Un *agent* est une entité logicielle ou matérielle qui interagit, via des modalités de perception et d'action, avec un environnement *partagé* avec d'autres agents, qui interagit avec les *autres agents* et dont le comportement peut être qualifié d'*autonome*. Un *système multi-agent* est l'ensemble des agents en interaction au sein de l'environnement partagé et montrant éventuellement une activité organisée.

Les applications caractéristiques de ce paradigme se situent dans au moins trois grandes classes de problématiques : résolution distribuée de problèmes avec des applications de supervision [17], [57] par exemple, simulation avec des applications de modélisation de phénomènes sociaux, [11] par exemple, et enfin des applications distribuées, décentralisées et ouvertes telles que le commerce électronique [19], les télécommunications [10].

2.2 La programmation par aspects

Si, la programmation par objets a constitué un progrès indéniable dans le développement logiciel, elle n'est cependant pas exempte de défauts. En particulier, l'usage des objets conduit au mélange et à la dispersion des propriétés transversales des applications. La concurrence, la persistance ou la communication distante sont trois exemples de propriétés transversales. De telles propriétés se trouvent mélangées avec le code métier et dispersées entre les différentes classes constituant une application donnée. De ce fait, le développement, la maintenance et l'évolution des applications se trouvent complexifiés.

En réponse à ce problème, la programmation par aspects (AOP, *Aspect-Oriented Programming*) a été introduite en 1997 [33]. Sans remettre en cause les fondations qui ont fait le succès de la technologie objet, l'AOP vise à séparer les définitions des propriétés transversales des logiciels. Chacune de ces propriétés, appelées *aspects*, est définie totalement et exclusivement dans un unique module. D'où une simplification du développement et donc moins de risque d'erreurs, puisque les développeurs peuvent se focaliser sur un seul problème à la fois. Une fois les différents aspects définis, ils sont assemblés pour construire l'application. Ce processus d'intégration, essentiellement automatique, est appelé *tissage* (weaving). Il consiste à lier les aspects avec le reste de l'application en des points particuliers appelés *points de jonction*. Ces points sont soit situés dans le code (définition d'une méthode, d'une classe, ...) soit dans le flot d'exécution (envoi de message, lecture d'un champ, ...).

Notons que les exemples les plus couramment rencontrés dans la littérature correspondent à des aspects techniques qui décrivent des éléments d'infrastructure (communication distante, transaction, ...). Cependant, la notion d'aspect est plus générale et couvre également les aspects métier. C'est le cas, par exemple, dans les travaux menés au laboratoire de recherche IBM Watson [26][54].

Un autre point à souligner est la généralisation de l'usage du paradigme aspect à tout le cycle de vie des

logiciels. A l'origine, la notion d'aspect a été introduite dans le contexte de la programmation. Néanmoins, son usage s'est généralisé pour couvrir, entre autre, l'analyse, la conception ou l'évolution des applications. Pour preuve, la série de conférences internationales dédiées à l'aspect est appelée *AOSD* pour *Aspect-Oriented Software Development* [30] [34].

Bien que relativement récente, la technologie des aspects commence à être utilisée pour des applications à grande échelle. Une des premières applications remonte à 1999 où la plate-forme à aspects AspectJ [31] a été utilisée dans le développement d'un serveur web de e-learning [29]. D'autres applications de diverses plate-formes à aspects ont suivi comme le démontrent les sessions sur les expériences industrielles dans la série de conférences AOSD [30] [34]. On apprend ainsi que des expérimentations sont menées dans les logiciels pour l'aviation [47] ou dans les EAI (Entreprise Application Integration) [46].

2.3 La programmation par composants

Le concept de composant est apparu en informatique pour répondre au problème de la crise du logiciel. Ce terme désigne l'accroissement exponentiel des moyens qui sont nécessaires au développement d'un logiciel, du fait de l'accroissement de leur complexité (richesse fonctionnelle) et de leur sophistication (par exemple le développement systématique d'interfaces utilisateurs graphiques). A titre d'exemple, le passage de la version 4 à la version 5 du traitement de texte WordPerfect a nécessité une somme de travail équivalent à 14 hommes/an. Le passage de la version 5 à la version 6 a nécessité l'équivalent de 250 hommes/an. Une solution à ce problème est de pouvoir systématiser la réutilisation d'éléments déjà existants dans le développement de nouveaux logiciels. Des études ont montré qu'en moyenne, 80% du code d'un logiciel est constitué de fonctionnalités standard. Seulement 20% du code correspond à des fonctions spécifiques, représentant la plus value de l'application. Un *composant* désigne une unité conçue pour être réutilisée dans le développement de logiciels. L'idéal est de disposer, sur étagère, des composants permettant de produire l'ensemble du code standard d'une application. Au delà d'une réduction des coûts, la réutilisation de composants permet d'augmenter la qualité des applications. Un composant ayant eu une certaine durée de vie, intégré avec succès dans de multiples applications et manipulé par une base d'utilisateurs importante, a moins de chance de souffrir d'erreurs de codage ou de conception qu'une production originale. Le concept de composant est applicable à différentes phases d'ingénierie : analyse du besoin (composants de cas d'utilisation), spécification (patrons d'analyse), conception (patrons de conception), codage (composants logiciels). Le processus d'ingénierie lui-même est défini sous forme de composants dans certaines méthodes (Open, Unified). Le développement à base de composants revient alors schématiquement à trois étapes : une étape de *sélection*, qui consiste à rechercher et proposer un ensemble de composants qui répondent au mieux à l'expression d'un besoin parmi une bibliothèque de composants disponibles ; une étape d'*adaptation*, qui consiste à configurer ou à spécialiser un composant afin qu'il réponde à des utilisations spécifiques dans une application donnée ; une étape d'*intégration*, qui consiste à assembler et connecter ensemble les composants afin que leur structure globale et leurs interactions réalisent l'application finale. Le travail de recherche sur le thème de l'ingénierie à base de composants de systèmes a pour objectif de proposer des modèles, des mécanismes et des outils qui permettent de mettre concrètement en œuvre une telle démarche. Dans la pratique industrielle, la réutilisation est encore essentiellement pratiquée au niveau des composants logiciels et repose sur des infrastructures qui assurent l'interopérabilité de composants distribués et hétérogènes grâce à des bus logiciels (Java Enterprise, CORBA(OMA), .NET). Elle commence à se développer au niveau des ateliers de génie logiciels par l'intégration de bibliothèques de patrons de conception. Elle devrait se généraliser rapidement à d'autres phases sous l'impulsion d'initiatives telles que MDA, une approche dirigée par les modèles promue par l'OMG (consortium d'industriels et de chercheurs, à l'origine de nombreuses technologies orientées objets standard telles que CORBA et UML). Ces approches visent à faire à graduellement reposer l'ingénierie à base de composants uniquement sur des modèles conceptuels, puis à obtenir les architectures de composants logiciels correspondantes par génération de code. L'objectif est de renforcer la

réutilisabilité des composants, en les exprimant de manière technologiquement neutre (car les composants logiciels doivent être réécrits à chaque saut technologique des plateformes d'implémentation). Une autre tendance induite par la démarche est de distinguer trois types d'acteurs dans l'ingénierie à base de composants : les fournisseurs de technologies, qui définissent les modèles de composants et les outils (chercheurs et industriels experts) ; les fournisseurs de composants, qui produisent des bibliothèques de composants à réutiliser (industriels utilisateurs des technologies orientées composants) ; les architectes, qui vont produire des applications pour les clients finaux en sélectionnant/adaptant/assemblant des composants (industriels utilisateurs de bibliothèques de composants). MDA est conçu par les premiers acteurs en direction des seconds. Pour la troisième catégorie d'acteurs est proposé le concept de langage de description d'architecture (ADL), domaine de recherche qui profite également de l'engouement pour les composants. L'objectif de ces langages est de permettre aux architectes d'exprimer comment un ensemble de composants peut être déployé, configuré et assemblé pour constituer une application (concepts qui ne sont pas ou peu couverts par les langages de conception ou de programmation actuels, car encore trop orientés objets et non spécifiquement orienté composant comme les ADLs).

2.4 Le contexte industriel

Actuellement le nombre d'ordinateurs, de plate-formes et d'applications informatiques ne cesse de croître. Internet et les différents réseaux de télécommunication ne font que renforcer et augmenter la complexité des systèmes informatiques. La coopération entre logiciels (composition de services), entre humains et logiciels (conception concurrente, travail collaboratif, commerce électronique) et coopération entre logiciels et systèmes mécaniques, robotiques (supervision d'ateliers, production manufacturière) sont au centre des préoccupations actuelles. De nouveaux outils informatiques doivent être développés afin de construire des architectures logicielles qui soient flexibles, qui intègrent la décentralisation et l'ouverture intrinsèque des applications, qui permettent la coopération de multiples services hétérogènes et autonomes.

Dans ce contexte, les Systèmes Multi-Agents (SMA) apparaissent comme des technologies adaptées pour prendre en compte ces caractéristiques. En effet ces systèmes sont constitués d'un réseau d'entités autonomes et coopérantes. Ceux-ci proposent de nombreux modèles que nous pouvons organiser selon quatre grandes familles, à savoir : Agents, Environnements, Interactions et Organisations. La réalisation d'une application s'appuyant sur ces technologies consiste à développer ou choisir dans des bibliothèques, un ou plusieurs modèles dans chacune de ces familles puis à les combiner. Ces choix de modèles ainsi que leurs combinaisons sont fortement dépendants de l'application.

La technologie des composants est née d'une demande de plus en plus forte de réutilisation du code lors de la conception d'applications informatiques. On a ainsi vu apparaître plusieurs propositions de langages faites par les grands groupes SUN, Microsoft, IBM, etc. L'idée principale est de s'inspirer de la conception des circuits ou de la mécanique pour lesquelles nous avons un catalogue de composants industriels que l'on assemble ou adapte en fonction des besoins du produit à réaliser. La technologie des composants veut promouvoir la définition de composants logiciels sur étagère et la conception d'application par assemblage de ceux-ci. L'intérêt de cette approche est de faciliter l'évolution du logiciel au cours de son cycle de vie, notamment avec la possibilité d'échanger des composants. D'une manière générale, le concepteur peut agir directement au niveau de l'assemblage des composants et non plus au niveau du code pour construire ou faire évoluer son application.

La troisième technologie qui nous intéresse est celle des aspects. Un aspect est une propriété qui coupe de façon transversale les différents modules d'un programme, par exemple la sécurité, la persistance, etc. L'idée de la programmation par aspect est d'extraire ces éléments de l'application. Les aspects sont ensuite cousus à la compilation ou à l'exécution dans le programme de base pour recréer une application complète.

Les systèmes multi-agents ou à base de composants sont, d'une façon générale, employés dans la conception de systèmes distribués. Leur intérêt est de permettre la conception d'un système logiciel en une organisation d'unités

logicielles relativement indépendantes et qui coopèrent et interagissent. Leurs avantages sont bien connus dans le domaine du génie logiciel ou de l'intelligence artificielle et on assiste à une multiplication des applications et des plate-formes.

Nous voulons proposer une infrastructure minimale, basée sur des standards (FIPA, XML, Java) et utilisable dans un contexte industriel (Eclipse) pour la réalisation de ces applications. Elle proposera aux développeurs une approche de haut niveau basée sur les composants, plus productive et plus sûre.

Les trois technologies évoquées sont complémentaires : une meilleure réutilisabilité et une architecture logicielle explicite avec les composants, la modularisation des propriétés transversales grâce aux aspects et la modélisation des systèmes complexes par les agents.

3 Synthèse des travaux

Les activités des quatre partenaires sont assez diverses mais recouvrent bien le champ d'étude envisagé à savoir les composants, les agents et les aspects. Nous présentons dans ce qui suit les travaux pertinents réalisés par les partenaires en 2003. Une synthèse de chacune des directions est faite et une synthèse globale est présentée dans la section 5. Nous discutons également certains travaux à l'intersection de plusieurs technologies. Pour plus de détails sur les travaux des participants le lecteur peut consulter les rapports d'activité [40]

3.1 Agents

3.1.1 EM Alès

L'axe développé à l'Ecole des Mines d'Alès dans le cadre de la thèse de Frédéric Souchon vise à renforcer la fiabilité des systèmes à base de composants ou multi-agents¹. Ces nouveaux paradigmes de développement ont pour caractéristique commune de nécessiter la coordination de l'activité d'entités logicielles indépendantes (développées séparément, distribuées sur différents serveurs, exécutées de manière concurrente et communicant de manière asynchrone). Dans ce contexte, nous étudions l'intégration de mécanismes de gestion d'exception qui permettent de signaler et de traiter les anomalies qui surviennent dans l'accomplissement de l'activité collective à laquelle un ensemble de composants ou d'agents contribuent (collaboration). Ces travaux contribuent au rapprochement des paradigmes de développement à base d'agents et de composants. L'introduction dans les modèles d'agents de bonnes pratiques du génie logiciel, héritées des langages à objets et qui prévalent dans les modèles de composants, tel le support de la gestion d'exceptions, est de nature à améliorer la qualité des applications développées avec des systèmes multi-agents, donc de favoriser leur utilisation à des fins industrielles (passage à l'échelle). Sur ce point, nos premiers résultats sont la proposition d'un modèle d'exécution d'agents intégrant un système de gestion d'exceptions décentralisé (prototypé pour le système multi-agents MadKit). Inversement, les modèles de composants gagnent en ouverture et autonomie (possibilité d'intégration, de remplacement, de suppression dynamique de composants) lorsqu'ils proposent des modèles d'exécution proches de ceux des agents (composants actifs, communication par messages abstraits échangés de manière asynchrone). Dans ce contexte, et malgré la flexibilité du modèle de composants utilisé, il faut veiller à ce que la fiabilité des applications continue à être assurée. Pour répondre à ce problème, nous avons adapté le modèle d'exécution et le système de gestion d'exception que nous avons conçu pour les systèmes multi-agents à la gestion d'un modèle de composants pilotés par messages (prototypé dans la plate-forme Jonas comme une extension du modèle de composants Mdb proposé par l'architecture J2EE et du protocole de messagerie asynchrone JMS). Ce dernier modèle peut être vu comme la base commune d'un modèle de composants ouverts ou d'un modèle d'agents intégrant un système de gestion d'exception proposé pour la plate-forme J2EE. Ces travaux ont donné lieu aux publications suivantes [52, 51].

¹Ce travail est mené en collaboration avec Christophe Dony, Pr., LIRMM, Université de Montpellier II.

3.1.2 EM Douai

L'équipe de Douai mène des travaux sur l'utilisation conjointe des aspects et des agents voir la section 3.4.

3.1.3 EM Nantes

L'équipe OBASCO de Nantes ne mène pas de travaux spécifiques sur les agents.

3.1.4 EM Saint-Etienne

Le développement d'un Système Multi-Agent (SMA) nécessite la prise en compte de plusieurs dimensions. L'approche Voyelles [18] (également désignée par l'abréviation AEIOU) distingue cinq dimensions : Agent, Environnement, Interaction, Organisation et Utilisateur. Chacune de ces dimensions participe à la modélisation d'un SMA en s'intéressant à l'architecture et aux connaissances des entités logicielles autonomes (agents), à la manière dont ils interagissent entre eux (Interaction), avec des utilisateurs humains (Utilisateur), avec un environnement extérieur (Environnement). La dernière dimension permet d'explicitier un ensemble de contraintes globales telles que normes, organisations influant sur le comportement des agents (Organisation). Pour chacune de ses dimensions, de nombreux modèles existent, adaptés à des contextes et des types d'applications spécifiques. Il existe, par exemple, plusieurs modèles d'interaction entre agents qui vont de l'expression d'un signal simple à l'émission et la compréhension d'actes de langages exprimant une sémantique plus complexe.

L'originalité des travaux menés à l'Ecole des Mines de Saint-Etienne est de distinguer au sein des modèles relatifs à ces cinq dimensions deux niveaux : (i) un niveau *global*, commun à l'ensemble des agents tel que, par exemple, langage de communication, organisation, ..., (ii) un niveau *local*, propre à chacun des agents visant à l'interprétation de ces modèles globaux au sein de l'agent. Nous avons ainsi proposé plusieurs modèles qui peuvent aborder à la fois le niveau global du SMA et sa prise en compte locale par les agents. Jusqu'ici, nos travaux se sont surtout focalisés sur les dimensions Agent, Interaction et Organisation d'un SMA :

- Dimension Agent : l'architecture d'agent STAx [1] est adaptée à la conception d'agent manipulant des informations temporelles. L'architecture TAG [16] apporte, en plus de la prise en compte d'informations temporelles, un premier support d'intégration pour des facettes locales selon la vision AEIOU.
- Dimension Interaction : nous nous sommes particulièrement intéressés aux interactions par actes de langage entre deux agents du système en proposant un langage de communication spécialisé sur l'expression de contraintes temporelles (TAFL [16]) puis par plusieurs travaux sur des protocoles d'interaction qui structurent les échanges de messages entre agents. Des travaux ont été menés sur l'introduction d'un serveur de protocoles [15] dans un SMA, ainsi que sur le raisonnement et le suivi local d'un protocole dans le cas de l'entrée ou la sortie d'un agent dans un SMA (SAMOS [56]) ou lors d'une négociation avec d'autres agents (NegF [12]).
- Dimension Organisation : nous avons proposé, pour modéliser la structure organisationnelle d'un SMA, deux formalismes : MOISE [25] et MOISE+ [27]. Le formalisme MOISE permet d'attribuer des rôles aux agents et des liens entre ces rôles, alors que MOISE+ étend ce premier formalisme pour y introduire des notions d'obligations et de permissions. Des travaux ont également été menés sur la modélisation d'un contrat sous cette même forme (obligations et permissions) en y incluant la notion de pénalité de rupture de contrat (ConF).

3.1.5 Synthèse

De nombreux environnements de développements² de SMA voient le jour actuellement dans le domaine (jade³ [2] et zeus⁴ – environnements conformes aux spécifications de la FIPA –, jack⁵ – basé sur le modèle BDI développé dans [41] –, AgentBuilder⁶ – basé sur le modèle AOP –, madkit⁷ – basé sur le modèle AGR –, etc). Ces environnements mettent en avant modularité et réutilisabilité. Ces propriétés n’apparaissent cependant qu’au niveau système via la proposition, selon les environnements, de modèles d’interaction, de comportements, d’organisation et peu au niveau agent. L’architecture de l’agent apparaît cependant comme le lieu où se cristallisent l’ensemble des choix de conception et d’implémentation. Les architectures proposées sont souvent dédiées à un mode de fonctionnement, un mode de représentation des connaissances. La généralité ne s’exprime qu’en terme de compétences de base des agents et non en terme d’agencement et d’activation de ces compétences.

Avec le développement de la notion de composant logiciel, se dégage aujourd’hui une volonté de proposer un ensemble de composants logiciels venant s’intégrer sur une architecture particulière, pour constituer un agent particulier répondant aux exigences de l’application [48, 44, 13, 3, 37, 42]. Devant une telle prolifération dont l’objectif est la réutilisabilité via la possibilité de composer “son” architecture d’agent, se pose un ensemble de questions sur la dimension, les fonctionnalités englobées dans chacun des composants, sur la manière de les composer, avec l’utilisation ou non de langages de description agent en très forte relation avec les langages de description d’architecture. Le deuxième point qui n’est pas abordé dans le domaine du multi-agent est la fiabilité, propriété au centre des recherches menées au sein de l’Ecole des Mines d’Alès (cf. section 3.1.1. L’étude des implications de la prise en compte de cette propriété dans la conception d’une architecture d’agent doit être menée de manière concomitante avec la définition des composants.

3.2 Aspects

3.2.1 EM Alès

L’équipe d’Alès ne mène pas de travaux spécifiques dans le domaine de la programmation par aspects.

3.2.2 EM Douai

Les travaux menés à Douai sur la programmation par aspects visent à définir un modèle d’aspects réutilisables et facilement intégrables. Ce travail ne s’arrête bien entendu pas à la simple définition d’un modèle, mais se poursuit jusqu’à son implantation. La plate-forme ainsi obtenue est ensuite expérimentée pour la construction d’applications à base d’aspects.

Différentes techniques peuvent être utilisées dans la réalisation de telles plate-formes [8]. Ces techniques doivent répondre aux questions de l’isolation des définitions des aspects et de leur tissage. Nous avons choisi de recourir à la *réflexion* [4][9]. La réflexion est la capacité d’un système à raisonner et à agir sur lui-même [49][35]. Plus concrètement, un logiciel réflexif peut modifier, non seulement les données qu’il manipule, mais également sa propre structure (le programme) et sa propre sémantique (l’interprète). Cette capacité appelée *intercession* nécessite de réifier les différents éléments du programme et de l’interprète sous forme d’objets appelée *méta-objets* [32].

²Nous utilisons ici de manière générique le terme environnement de développement pour désigner ces différents outils.

³Java Agent Development Framework

⁴<http://www.labs.bt.com/projects/agents.htm>

⁵<http://www.agent-software.com.au>

⁶<http://www.agentbuilder.com>

⁷<http://www.madkit.org>

Dans le contexte de la réflexion, un aspect est défini sous la forme d'un ensemble de méta-objets. Le tissage est quant à lui réalisée grâce à l'intercession. Par exemple, en modifiant la sémantique des accès aux champs, il est possible d'intercepter les lectures/écritures des champs des objets pour les synchroniser. De plus, comme la réflexion permet de modifier le programme en cours d'exécution, il est possible d'opérer la modification précédente dynamiquement. De manière plus générale, la réflexion permet d'introduire ou de retirer les aspects durant l'exécution. Cette faculté est très intéressante dans les systèmes critiques qui doivent être maintenus sans pour autant être stoppés. C'est le cas par exemple des logiciels de télécommunications (centraux téléphoniques, satellites, ...), de contrôle de processus industriels ou de production d'énergie.

Un autre intérêt du recours à la réflexion est de ramener le tissage à un problème mieux connu. En effet, comme les aspects sont représentés sous forme de (méta-)objets, tisser les aspects revient à assembler des objets.

Le troisième intérêt de la réflexion est qu'elle pousse à l'écriture d'aspects génériques et donc réutilisables. Le protocole des méta-objets (MOP) constituants les aspects, définit les familles de points de jonction où peuvent intervenir les aspects (créations d'objets, envoi de message, ...). Nous avons exploité ce cadre pour définir un modèle d'aspects réutilisables [4][6][9]. Ce modèle relâche le couplage entre les aspects et leur contexte d'exécution. Un aspect peut être défini indépendamment de tout contexte et donc, sans préciser les points de jonction où il intervient. Le développeur de l'aspect ne définit que les familles de points de jonction où l'aspect est susceptible d'intervenir. L'aspect ainsi obtenu est générique. Ce n'est qu'au moment de la préparation du tissage que l'aspect est couplé au contexte d'une application. En effet, l'intégrateur doit fournir pour chaque aspect les points de jonction où il doit s'assembler avec le reste de l'application.

Le modèle d'aspect que nous avons défini ainsi que la plate-forme correspondante servent de support pour différents travaux. Il s'agit d'une part d'étudier les différents problèmes soulevés par la programmation par aspects et son utilisation. Le DEA de Rabih Nassrallah s'inscrit dans ce cadre. Il a consisté à définir la communication distante des services web sous forme d'un aspect réutilisable [38]. Cet aspect peut être exploité dans différentes applications qui n'étaient pas initialement prévues pour être réparties. Le tissage de cet aspect (après paramétrage) dans une application lui confère la capacité de communication distante via le protocole SOAP des services web. Les éléments de l'application en question peuvent alors être déployés sur des machines différentes.

D'autres travaux concernent la fertilisation croisée de la programmation par aspects avec d'autres domaines de recherche. C'est le cas de nos travaux sur l'intégration des aspects avec les composants logiciels (voir la section 3.6) et de l'application des aspects aux SMA (voir la section 3.4).

3.2.3 EM Nantes

Un objectif des travaux de l'équipe de Nantes autour des aspects est de définir un modèle formel pour la programmation par aspects basé sur les concepts d'événement, de trace et de moniteur. Il s'agit également d'implémenter le langage correspondant en utilisant la réflexion et les techniques d'analyse et de transformation de programmes. Le modèle EAOP est une approche dynamique pour la programmation par aspects, il permet un pouvoir d'expression important par l'utilisation de patrons d'événements.

Le modèle propose deux variantes importantes : statique et dynamique. Le modèle statique est plus restrictif mais cela autorise l'utilisation de contrôles plus efficaces des conflits d'interaction entre aspects. La réutilisation des aspects a été accrue par un enrichissement de leur interface. Le modèle dynamique repose sur une implémentation en Java, cette implémentation a été optimisée par l'utilisation d'une technologie avancée (passage de continuation). Ce prototype permet de démontrer que différents mécanismes comme la composition d'aspects, l'instanciation d'aspect ou la visibilité des aspects peuvent être traités de façon uniforme.

L'utilisation de la réflexivité (Reflex) comme point de départ pour la programmation par aspects a été illustrée, ce qui conduit à une autre approche permettant une meilleure comparaison des performances et de la puissance d'expression. Les aspects ont été appliqués dans des applications non triviales comme l'évolution du noyau Linux

pour supporter l'ordonnanceur Bossa, ou des stratégies dynamiques pour l'utilisation de cache Web sans interruption du traitement des requêtes.

3.2.4 EM Saint-Etienne

L'équipe de Saint-Etienne n'a pas d'activité dans le domaine des aspects.

3.2.5 Synthèse

On peut noter deux principales approches de la programmation par aspects : statique et dynamique suivant le moment où ceux-ci sont cousus avec le programme principal. Il faut également noter que deux approches sont décrites pour les introduire : la transformation de programme ou la réflexivité. Cette dernière technique a été expérimenté par les équipes de Douai et de Nantes et consitue donc un point commun important.

3.3 Composants

3.3.1 EM Alès

Le groupe d'Alès s'intéresse à la réutilisation de composants en se basant sur l'assemblage dirigé par les modèles. Après avoir essentiellement consisté à intégrer, au niveau implémentation, des composants logiciels interopérables, à l'aide d'infrastructures telles qu'OMA (Corba), J2EE et plus récemment les web services, l'ingénierie à base de composants des logiciels et des systèmes d'information s'enrichit de démarches dirigées par les modèles. Ces démarches sont fondées sur un constat simple : les composants logiciels ne tiennent pas leurs promesses en termes de réutilisation. Les infrastructures qui permettent l'intégration des composants logiciels ne sont pas matures : elles subissent des évolutions rapides qui obligent à faire migrer régulièrement le code des composants vers ces infrastructures. Seuls les modèles conceptuels de ces composants, produits lors des étapes de spécification ou de conception, constituent des productions pérennes et capitalisables. Ce sont donc ces modèles conceptuels qui doivent constituer les véritables éléments réutilisables d'un logiciel, c'est-à-dire les véritables composants.

Ainsi, les approches dirigées par les modèles proposent-elles de réaliser l'intégration des logiciels non plus au niveau implémentation, grâce à des composants logiciels interopérables, mais au niveau conceptuel, par l'intégration de composants conceptuels, décrits dans un langage de modélisation standard et technologiquement neutre tel qu'UML. Dans ce type d'approche, les composants logiciels implémentant ces composants conceptuels sont obtenus à l'aide de générateurs de code spécifiques à l'infrastructure cible, ce qui permet de faire migrer à moindre coût les composants.

Dans ce contexte, nous avons étudié la création de composants conceptuels pour et par la réutilisation. La relation d'association est l'un des concepts clés de la mise en oeuvre pratique de cette réutilisation : elle permet de lier des classes provenant de différentes modèles, en exprimant les rôles relatifs qu'elles jouent entre elles, et de créer ainsi de nouveaux modèles, par assemblage de modèles existants. Nous avons identifié les principales propriétés des relations d'association au regard de leur impact sur les mécanismes de définition puis d'utilisation de composants conceptuels. Nous en avons déduit les caractéristiques nécessaires à un modèle de relations d'association pour faciliter le découplage des composants (ce qui permet de produire des composants réutilisables) puis leur assemblage (ce qui permet de réutiliser effectivement des composants). UML, en tant que standard pour les langages de modélisation, et MDA, en tant que futur standard de démarche dirigée par les modèles proposé par l'OMG, nous servent de cadre pratique d'illustration.

Ces travaux constituent une première étape pour atteindre un objectif plus ambitieux : proposer un modèle de composants intelligents, capables, dans un milieu ouvert, de s'organiser (s'assembler et se déployer) automatiquement. Une étape ultérieure de ces travaux consistera à embarquer dans les modèles des composants des

informations sémantiques sur les collaborations qu'ils souhaitent/peuvent réaliser avec les autres composants du système (par exemple sous la forme de diagrammes d'état ou de collaboration). Ces informations sémantiques auront pour objectif d'analyser automatiquement non seulement la compatibilité syntaxique de deux composants (ce qui était l'objet des travaux présentés ci-dessus) mais aussi, autant que possible, leur compatibilité sémantique. Ces travaux ont fait l'objet des publications [36, 55].

3.3.2 EM Douai

Les travaux que nous menons au sein de l'École des Mines de Douai visent à définir un modèle de composants (concepts, principes, implémentation) qui réponde le mieux aux définitions les plus « standard » posées par cette technologie [53]. A cet effet, nous avons étudié différents modèles existants. Notre objectif est de projeter les définitions d'un modèle de composant idéal sur ces modèles existants pour leur proposer soit des extensions possibles leur permettant de mieux refléter les définitions standards soit de les doter de certaines fonctions qui sont liées à d'autres paradigmes (principalement les aspects et la représentation multiple).

Le premier modèle que nous avons étudié est le modèle de composants Corba (CCM) défini par le consortium OMG (<http://www.omg.org/>). Ce dernier possède un ensemble de propriétés lui permettant d'être le plus proche, à notre avis, des définitions « standard » des composants logiciels. Néanmoins, l'une des faiblesses de ce modèle concerne les possibilités offertes pour l'assemblage de composants. Ce constat, partagé par d'autres groupes de recherche comme ACCORD (<http://www.infres.enst.fr/projets/accord/>), a poussé ces groupes à s'intéresser à la notion de connecteur dans le modèle CCM. Cependant, nous pensons qu'une des caractéristiques fondamentales liées à l'assemblage de composants est, aussi, la capacité de créer des composants logiciels complexes (appelés composites) à partir de l'assemblage d'autres composants plus élémentaires (des sous-composants). Les composants composites doivent être liés aux sous-composants par une relation particulière : la composition. La gestion de ce type de lien nécessite la gestion de certaines caractéristiques induites telles que les liens existentiels entre le composite et les sous-composants, la multiplicité et le partage des sous-composants ainsi que la propagation des requêtes entre le composite et les sous-composants.

Salah Hadjab a concrétisé, pendant son DEA, cette relation de composition sous la forme d'une extension du modèle CCM [24]. Cette extension a été réalisée sur la base de la plate-forme OpenCCM, implantation de CCM réalisée par le consortium ObjectWeb. L'introduction de la relation de composition dans CCM s'est traduite par la définition du langage de description de composants baptisé IDL3+. Il s'agit d'une extension de IDL3. IDL3+ enrichie IDL3 par l'ajout du concept de composite et ainsi de tous les mécanismes nécessaires à la gestion de la relation de composition dans le modèle de composants CCM.

Nous avons également démarré l'exploration du modèle de composants Fractal défini par le consortium ObjectWeb (<http://fractal.objectweb.org/>). Comme CCM, Fractal est un modèle de composants qui se veut indépendant de tout langage. Fractal est cependant plus riche que CCM. En effet, il introduit notamment les notions de composant composite (composant construit à partir d'autres composants) et le concept de contrôleur de composant (élément de composant permettant d'inspecter voire de modifier le composant). Par ailleurs, Fractal nécessite une infrastructure plus légère que CCM (ce dernier est destiné à fonctionner au dessus de CORBA). Dans le but d'utiliser Fractal avec les outils déjà développés à Douai, nous avons démarré la projection de ce modèle sur Squeak [28][23][14]. Ce dernier étant une implantation libre (open source) du langage à objets Smalltalk [22].

3.3.3 EM Nantes

L'équipe INRIA/EMN OBASCO s'intéresse aux composants et a lancé un travail de réflexion et de synthèse sur cette approche qui est à la base du projet OBASCO. Nous avons conduit une analyse ainsi que diverses expérimentations sur les langages ou modèles de composants. En particulier les langages Fractal et ArchJava

ainsi que des propositions relevant plus des langages de description d'architecture (Wright). Un point qui semble important dans les composants modernes est de permettre l'expression du comportement dynamique. Cet ajout a comme objectif principal de renforcer la précision et la sécurité du modèle à composants. Avec l'expression de la dynamique nous pouvons vérifier si un assemblage de composants explicite ou non les comportements attendus ou encore contrôler que le changement d'un composant par un autre n'est pas *a priori* illégal. Les deux approches qui ont été étudiées reposent sur l'utilisation de machines à états finis, on peut généraliser les deux en utilisant une notion de système symbolique proche des diagrammes d'états d'UML. Des travaux dans le cadre de la thèse de A. Farias ont montré que l'on pouvait appliquer une approche issue des travaux de O. Nierstrasz pour décider de la compatibilité des protocoles [39]. Ce travail a été implémenté en Java en utilisant le canevas pour la compilation d'Eclipse. La thèse sur ce sujet a été soutenue fin 2003 [20].

Nous avons également étudié le mode de communication asynchrone en distinguant réception et activation des messages. Nous avons donc étendu nos travaux précédents dans cette voie en réutilisant les principes des systèmes symboliques. Ce travail a été fait dans le cadre du Master EMOOSE de Michael Xu [58]. Nous avons donc une approche rigoureuse pour analyser un système de composants asynchrones, une expérimentation de codage en ProActive (sur-langage de Java) a été faite. Finalement nous avons défini et implémenté un algorithme d'analyse qui permet d'analyser et de calculer le comportement des boîtes à lettres des composants. Ce travail a été publié à la conférence DOA'2003 [45].

Le travail du stage de DEA de Sébastien Pavel se concentre sur les possibilités d'appliquer une approche de construction du logiciels, appelé *Ligne de Produits* à un modèle de programmation par composants, ArchJava. Dans l'industrie, les lignes de produits sont utilisées depuis longtemps. L'idée est d'avoir un modèle de produit très général et à partir de celui-ci on va en décliner des types de produits finaux. Même si les avantages de cette approche dans le domaine du logiciel sont évidents, les travaux de recherche sont loin d'être achevés. Une cause possible est le fait que les produits d'une ligne reposent sur des composants logiciels et les modèles de composants sont loin d'être parfaits. Malheureusement le modèle d'ArchJava, même si il est facile à utiliser, ne permet pas d'implémenter simplement et directement une ligne de produits. Ceci est dû au fait qu'ArchJava essaie à tout prix de garder l'intégrité des communications, les possibilités de configuration et de spécialisation des composants, surtout dynamiques, sont très limitées. Techniquement il n'est pas possible d'avoir des instances de composants en arguments des communications. Ceci oblige donc à avoir recours à des astuces de programmation peu exploitables dans un contexte de Génie Logiciel.

L'adaptation classique des composants consiste essentiellement en la modification de valeurs paramètres. Toutefois dans les applications actuelles nous avons besoin de plus de souplesse (dynamisme des liens de communications, par exemple) ou d'efficacité. Les techniques classiques de spécialisation ou d'évaluation partielle ne s'appliquent pas bien dans le contexte des composants boîte-noire. Le travail de Master EMOOSE (2003) de Diego de Sogos [50] a consisté en une expérimentation de générateur de composant contenant une description des opportunités de spécialisation. Pour éviter de casser le principe de la boîte-noire, le générateur de composant va produire des versions spécialisées du composant en tenant compte des informations d'adaptation.

3.3.4 EM Saint-Etienne

Le département SMA de l'Ecole des Mines de Saint-Etienne s'est intéressé aux modèles de composants dans l'objectif de concevoir des agents par assemblage de composants. Les architectures d'agents utilisées jusqu'ici étaient souvent très fortement marquées par l'utilisation d'un modèle particulier et rarement compatibles pour l'utilisation conjointe avec d'autres modèles. Un premier travail de spécification des besoins a été effectué pour déterminer les caractéristiques nécessaires à un modèle de composant pour la construction d'un agent :

- Modularité, réutilisabilité, interchangeabilité : un composant implémente une fonctionnalité clairement identifiée d'un agent. Une fonctionnalité pouvant être implémentée de différentes manières, elle peut être mise

en œuvre par différents composants. Le concepteur a ainsi le choix entre plusieurs composants, sans que ce choix n'ait d'influence sur le contenu des autres composants de l'agent.

- Modèle d'interaction entre composants : les composants doivent être en mesure d'interagir dans deux cas distincts : (i) demande d'information : un composant requiert une information contenue dans un autre composant ; (ii) contrôle : un composant active l'exécution d'un service encapsulé par un autre composant.
- Flexibilité des interactions entre composants : les interactions entre composants doivent être suffisamment flexibles pour être modifiées après la conception d'un agent et même pendant son exécution.
- La notion de distribution, présente dans de nombreux modèles de composants, n'est pas nécessaire dans notre cas car un agent est vu comme une entité élémentaire et indivisible du SMA.

Cette spécification générale des besoins a conduit au développement d'un prototype de modèle de composant pour la construction d'agents contenant :

- un modèle de composant décrivant ses rôles et les événements qu'il peut émettre ou traiter ;
- une structure pour les événements échangés ;
- une infrastructure support, appelée le noyau de l'agent (AgentCore), sur laquelle se connectent les composants et qui gère le routage des événements.

Un travail de validation et de consolidation de ce modèle est en cours à la fois d'un point de vue théorique (comparaison avec des modèles existants) et pratique (utilisation du modèle pour développer certains composants, agents et SMA simples).

3.3.5 En synthèse

Les travaux réalisés par les différentes équipes membres du projet autour de la notion de composant sont nombreux et variés.

Ils portent tout d'abord sur un des aspects fondateurs de la notion de composant : leur assemblage à des fins de construction d'applications par réutilisation (connexion, association, composition) et touchent à la fois aux aspects statique et dynamique de la description des composants. Une des originalités de l'équipe projet sur ce point est d'avoir étudié ces aspects selon deux démarches :

- une démarche descendante « des composants vers leurs applications » : en se posant la question des qualités que devrait posséder un modèle de composants minimum, ou idéal, afin de remplir au mieux ces objectifs de réutilisation. L'ensemble de ces travaux a donné lieu à des états de l'art et à de nombreuses expérimentations.
- une démarche ascendante « d'une application concrète vers la définition d'un modèle de composant » : en se posant la question des qualités attendues d'un modèle de composants qui permettraient, par assemblage, de réaliser le cœur du comportement d'un agent. Ces travaux ont donné lieu, notamment, à l'expression de besoins qui devraient être satisfaits par le modèle de composant adopté ainsi qu'à des études de modèles existants.

Ces deux démarches sont complémentaires et s'enrichissent l'une l'autre. Elles convergent vers un objectif unique : la définition d'un modèle de composants commun. Dans ce contexte, une perspective pour les travaux de l'équipe projet est de poursuivre le rapprochement des propositions expérimentées et des besoins recensés pour aboutir à la définition d'un modèle de composants minimal et commun, qui permettrait tout à la fois, de poursuivre les expérimentations sur les composants et de satisfaire aux exigences de l'application pressentie. Les qualités attendues de ce modèle sont en cours de définition par l'équipe. Les choix technologiques pour l'implémentation de ce modèle sont, eux aussi, en cours de définition (étude de Fractal, ArchJava et Reflex).

Les travaux de l'équipe projet sur le thème des composants apportent aussi des éclairages sur des aspects plus « pointus » des composants : composants et points de vue, composants et asynchronisme, fiabilité dans les collaborations de composants logiciels asynchrones, composants et lignes de produits, composants et spécialisation, composants « intelligents ». Chacune de ces thématiques d'étude pourra, dans une deuxième étape du projet,

donner lieu à des ajouts au modèle de composants commun et ainsi enrichir ses possibilités d'applications industrielles.

3.4 Agents et aspects

Une partie du travail de l'équipe de Douai est d'étudier les bénéfices de l'utilisation conjointe de l'AOP et des SMA. Cette relation semble être à la fois complexe et riche. Un travail de DEA de Romain Robbes [43] effectué à Douai en collaboration avec l'université de Caen a permis d'identifier différentes facettes et en particulier :

- L'AOP dans l'implémentation des SMA : cette facette de la relation AOP-SMA s'intéresse à l'utilisation de l'AOP dans l'implémentation des infrastructures des SMAs, où comment séparer au mieux les diverses fonctionnalités des plate-formes à agents. Parmi les aspects des SMA déjà identifiés, nous pouvons citer la recherche d'accointances et la gestion de l'envoi de message.
- L'AOP au niveau de la conception des SMA : il s'agit ici d'utiliser les techniques de l'AOP pour favoriser la conception de SMA, en favorisant le partage et l'abstraction des rôles et des responsabilités que les agents doivent occuper. Dans le cadre du modèle Aalaadin [21], l'analyse révèle que certains aspects (au sens AOP = propriétés transversales) ne sont pas clairement explicités. C'est le cas par exemple des contraintes sur les agents qui peuvent endosser des rôles définis dans différents groupes. Les protocoles de communication (au sens séquence et langage) sont un autre exemple de telles propriétés.

3.5 Agents et composants

Les travaux réalisés par Saint Etienne, mais c'est une observation faite également par d'autres ([37]), montrent que la notion de composant permet une définition plus rationnelle et plus réutilisable des agents.

Les travaux réalisés par Alès sur les plateformes SMA et les plateformes à composants montrent que les deux types de plateformes présentent des similitudes qui permettent de transposer facilement des résultats d'un type de plateforme vers l'autre. Ainsi, le système de gestion d'exception, conçu pour la plateforme à agents MadKit a pu être transposé « à l'identique » pour les composants asynchrones de la plateforme J2EE.

3.6 Composants et aspects

La séparation des aspects et les composants logiciels sont tous les deux des paradigmes prometteurs qui favorisent la réutilisation et simplifient le développement logiciel. Cependant, différentes questions restent ouvertes. Parmi elles : Comment intégrer la représentation des aspects dans les composants logiciels ? Quelle est la bonne granularité des composants ? Quels mécanismes devraient être utilisés pour construire des applications distribuées à base de composants ? Comment gérer les interactions et chevauchements entre aspects ?

Ces différentes questions sont abordées dans la thèse de Houssam Fakhri à Douai. L'objectif à long terme est d'aboutir à un paradigme de programmation fusionnant l'utilisation des composants et les aspects et ce, dans les systèmes distribués en particulier.

Le modèle EAOP a été testé sur une application de type commerce en ligne dans le contexte du projet Européen EASYCOMP. Concrètement EAOP a été intégré avec le canevas Vienna Component Framework (VCF) développé à l'université technique de Vienne en Autriche. Cette intégration permet de définir des aspects dynamiques pour les propositions industrielles de type EJB et .NET. Finalement une entreprise allemande H.E.I GmbH a développé une application de commerce en ligne en utilisant VCF et EAOP. Cette étude de cas permet l'introduction d'aspects gérant des règles métiers pour les politiques de rabais commerciaux dans une application existante.

4 Logiciels

4.1 EM Alès

Des démonstrations ont été réalisées dans le cadre du projet : Sage for MadKit et Sage for Jonas. Elles sont accessibles en ligne à l'adresse <http://www.lgi2p.ema.fr/~fsouchon/sage.html>.

4.2 EM Douai

MetaclassTalk (<http://csl.ensm-douai.fr/MetaclassTalk>) est une extension réflexive du langage à objets Small-talk. Cette extension permet d'ouvrir d'avantage le langage en le dotant de méta-classes explicites et d'un protocole de méta-objets (MOP, Meta-Object Protocol). Le protocole à méta-objets permet de contrôler l'exécution des programmes en permettant de manipuler à la fois la structure et le comportement des objets. Le contrôle de la structure des objets permet aux développeurs de définir les règles d'allocation de la mémoire et des accès en lecture/écriture aux champs. Quand au contrôle du comportement, il offre la possibilité de modifier les règles d'envoi/réception des messages, la recherche de méthodes et leur exécution. En donnant ces différents levier de contrôle, MetaclassTalk constitue un laboratoire pour expérimenter différents paradigmes de programmation [4]. Nous l'avons notamment exploité pour la mise en œuvre de la programmation par aspects [9] et l'introduction de l'héritage à base de mixins [7] [5].

MetaclassTalk a servi de base pour les prototypes réalisés dans le cadre de stages de DEA. Ainsi, Romain Robbes a développé une plateforme à agents en utilisant le support de la programmation par aspects fourni par MetaclassTalk [43]. Différents éléments de l'infrastructure tel que la communication asynchrone entre agents ont été implantés comme des aspects pouvant être modifiés ou remplacés. Rabih Nassrallah a quant à lui implanté, en tant qu'aspect, le support communication distante SOAP pour les services web [38]. Ainsi, une application peut être écrite et testée avant de la rendre répartie en y intégrant l'aspect pré-cité.

Un autre prototype développé dans le cadre du stage DEA de Salah Hadjab [24] est quant à lui indépendant de MetaclassTalk. Il s'agit d'une extension de la plate-forme OpenCCM, implantation de CCM réalisée par le consortium ObjectWeb. Cette extension nommée IDL3+, enrichie IDL3 par l'ajout du concept de composite.

4.3 EM Nantes

Ptidej (Pattern Trace Identification, Detection, and Enhancement in Java) est un ensemble d'outils qui démontre l'intérêt de la méta-modélisation et de la programmation par contrainte pour détecter l'application de patrons de conception dans du code. Les outils Ptidej www.emn.fr/x-info/ptidej implémentent ces modèles et algorithmes qui ont été intégrés dans l'environnement Eclipse pour Java. Ils incluent le méta-modèle PADL, des outils d'analyses statiques et dynamiques : Introspector and Caffeine. Un solveur de contrainte Ptidej solver, a été dérivé du résolveur de contrainte avec explication PaLM.

Reflex est une extension réflexive et ouverte pour Java. Il permet deux niveaux d'approches de la réflexion : un niveau simple pour les utilisateurs plutôt naïfs dans le domaine et un niveau avancé pour les architectes logiciels confirmés. Le prototype de Reflex est disponible à <http://www.emn.fr/x-info/reflex>.

Le modèle EAOP a été implémenté en Java, il permet soit des coutures d'aspect au niveau source code soit au niveau du code interprétable Java. La distribution des outils est disponible publiquement à l'adresse :

<http://www.emn.fr/x-info/eaop>.

Pierre-Charles David a proposé un langage d'architecture alternatif à celui par défaut proposé pour Fractal. Bien qu'un peu moins puissant, ce langage simplifié est beaucoup plus simple d'utilisation et il est également plus facile à lire. Cet outil est disponible publiquement sur le site ObjectWeb :

<http://fractal.objectweb.org/tutorials/simple-adl/index.html>.

4.4 EM Saint-Etienne

Le projet MAST (Multi-Agent System Toolkit) a pour objectif la réalisation d'un environnement de développement de systèmes multi-agents. Cet environnement assiste le concepteur d'une application multi-agent en lui fournissant des modèles génériques qu'il doit pouvoir combiner. L'environnement MAST est composé de cinq parties distinctes :

- *GeMas* est une bibliothèque de composants génériques, utilisables soit pour la conception globale du système (par exemple un modèle de structure organisationnelle) soit pour l'assemblage au sein d'un agent (par exemple, un composant lui permettant d'envoyer et de recevoir des messages).
- *MeMas* regroupe plusieurs outils graphiques de manipulation des modèles de la bibliothèque GeMas pour faciliter l'utilisation et le paramétrage de ces composants.
- *DeMas* est une plate-forme d'interaction qui assure le routage de messages entre agents.
- *AdMas* contient des outils d'administration et de déploiement d'une application multi-agent. Des schémas de déploiement peuvent être configurés à l'aide d'interfaces dédiées. D'autres outils permettent de superviser et d'administrer l'exécution d'une application.
- *AMas* est la couche la plus haute contenant l'ensemble des applications développées avec MAST.

Un modèle de composant a été introduit dans l'environnement MAST pour permettre l'utilisation conjointe de plusieurs modèles de la bibliothèque GeMas. Les outils de conception, de déploiement et de supervision ont été adaptés à ce modèle et des premières applications très simples ont pu être créées et exécutées.

5 Le modèle commun

Le modèle de composants pour la construction d'agents qu'utilise l'équipe SIMMO présente des spécificités que nous nous attacherons à préciser dans un premier temps. Il s'agit d'élaborer une spécification des besoins, une première approche de cette spécification informelle est la suivante.

Le modèle des agents de l'équipe SIMMO est organisé autour d'un BUS logiciel appelé *AgentCore*. Chaque agent est vu comme un BUS logiciel sur lequel sont greffés des composants dédiés à des fonctionnalités spécialisées (par exemple l'encapsulation de connaissances sur les agents du système, la gestion des conversations entre agents ou l'observation du flux des messages sur le BUS). Ces composants peuvent être inter-dépendants et interagissent soit par des requêtes d'échanges d'information ou par des requêtes de contrôle. Ce mode de communication par événements est asynchrone et précise si une réponse est attendue ou non. Un mécanisme de consommation des événements apporte une certaine flexibilité aux connexions entre composants, permettant ainsi l'interception et la supervision d'événements. Le besoin primordial qui est satisfait par ce modèle est l'inter-changeabilité des composants. La notion de rôle de composant décrit une catégorie de composants de manière abstraite et autorise la substitution d'un composant par d'autres composants de même catégorie. Enfin, chaque composant a la possibilité d'être associé à un processus indépendant.

Chaque agent est situé sur un seul site (le modèle de composant ne permet pas la distribution des composants). La composition de chaque agent est plutôt statique et n'évolue pas dynamiquement. Parmi les besoins non satisfaits par ce modèle mais pourtant requis pour la construction d'un agent, on trouve : (i) la définition de composants composites ; (ii) un traitement explicite des erreurs (gestion d'exceptions) ; (iii) la mise en place d'un véritable cycle de vie des composants (pour le moment, seule une notion sommaire de cycle de vie d'un composant est présente).

L'état actuel de nos travaux communs montre une orientation vers un modèle de composant *idéal*. Comme nous le verrons par la suite il ne sera pas forcément implémenté du premier coup mais expérimenté sur un langage existant disposant d'un support reconnu.

Nous privilégions une approche minimale *i.e.* un ensemble minimal de fonctionnalités essentielles et cohérentes. Toute fonctionnalité qui pourra être obtenue comme combinaison d'autres ne sera pas considérée dans le noyau minimal. L'ajout d'extensions pourra se faire de façon privilégiée par les aspects, mais la réflexion et les transformations de programme sont des outils pris en considération.

Le modèle ou langage de programmation doit être généraliste et en particulier prendre en compte la concurrence possible entre composants. Certains aspects comme la distribution des composants ne nous intéressent pas directement.

Une tendance récente est de voir les modèles de composants comme des sur-couches du modèle de la programmation à objets. La couche des composants et de l'architecture doit proposer des moyens de calcul pour permettre de la dynamique des configurations. Le langage d'architecture doit proposer une notion de colle permettant des connections simples, des conditions, de la dynamique, de l'expressivité pour les calculs, etc. Une question qui n'est pas encore résolue est de savoir comment interagissent précisément ces deux couches. Nos travaux devraient nous donner une réponse utile à cette question, cette question n'est pas anodine puisqu'elle touche à la fois à la puissance du langage mais aussi à la sécurité des programmes écrits.

Il est courant depuis les travaux avancés des langages de programmation de considérer les composants comme des citoyens de première classe du langage. Les composants sont vus comme des valeurs typées construites à partir d'un générateur de composant. Ceci a des conséquences importantes en rapport avec la mobilité/migration des composants et donc, là encore, au niveau de la sécurité des programmes.

Les modèles ou langages proposés aujourd'hui doivent proposer la composition de composants, on parle aussi de composants hiérarchiques. Chaque composant doit au moins disposer d'une description des services offerts ou requis, ainsi que du mode de communication synchrone et asynchrone.

La notion de connecteur telle qu'on la trouve dans certains langages n'est pas indispensable puisqu'elle peut être dérivée des composants à condition que ceux-ci expriment une notion d'interface de comportement dynamique. Une originalité qui nous intéresse est la description explicite du comportement dynamique des composants. Cette tendance va apparaître de façon plus forte dans les propositions futures de langage de composants. Une raison à cela est un besoin de sécurité, une autre est due à l'absence de connecteur prédéfinie. Il s'agit d'explicitier le protocole d'utilisation des composants, les enchaînements d'interactions qui sont autorisés. De nombreux langages et travaux existent dans ce domaine nous ne nous étendrons pas dans l'immédiat sur ce sujet.

Une originalité des composants est d'intégrer l'histoire de sa production jusqu'à sa mise en service, ceci afin d'être plus pertinent au niveau des adaptations. Cette notion de cycle de vie d'un composant renforce le besoin d'exprimer des comportements dynamiques dans un composant.

Des notions plus avancées nous intéressent pour les composants. La notion d'aspect semblent être une réponse à des problèmes récurrents concernant la description et l'intégration de propriétés transversales à une application. L'introduction des aspects améliore la modularité et la lisibilité des systèmes.

Le besoin de changer ou d'inter-changer des parties d'un système oblige à proposer des notions de compatibilité des composants. Une notion simple est celle associée aux interfaces, la notion est beaucoup moins évidente quand on considère des protocoles explicites dans les composants.

Une préoccupation fondamentale pour l'utilisateur final est tout ce qui concerne les propriétés de qualité de service. On veut pouvoir exprimer des propriétés générales mais aussi pouvoir les vérifier (de préférence statiquement quand cela est possible). Dans le cas où cette vérification n'est pas possible un mécanisme de forçage à base d'aspect peut être un bon compromis entre sécurité et efficacité.

La réflexivité est connue pour autoriser une meilleure compréhension et évolution des modèles proposés, une considération (secondaire) est de l'utiliser pour ouvrir notre modèle de composant.

Face à ces besoins nos analyses semblent concorder sur plusieurs points. Pour être efficace à court terme, opportun et démonstratif, le choix d'un langage comme Fractal 2.0 nous semble judicieux. Nous y trouvons un

langage hiérarchique avec déclinaison en Java mais aussi en C. Il propose des facilités d'introspection et de la reconfiguration dynamique d'architecture (création et assemblage dynamique de composant). Il dispose d'une notion de couche où sont bien identifiés les différents niveaux du langage (objets, composants, etc). Le langage n'a pas de connecteur prédéfini et il propose une notion de cycle de vie rudimentaire. Il ne propose pas de mode de communication asynchrone et un composant n'est pas nécessairement un processus.

Un aspect important est le support ObjectWeb, qui est un *consortium* créé en 1999 par Bull, Telecom R&D et l'INRIA pour promouvoir les systèmes distribués, ouverts et adaptables. Depuis 2002, c'est une organisation internationale. Deux centres (Nantes et Douai) conduisent déjà des travaux en utilisant ce langage.

Bien que nous estimons que Fractal semble être un candidat très sérieux, nous souhaitons nous faire une opinion plus précise sur d'autres propositions comme par exemple ArchJava. Une fois ce choix fait nous enrichirons le langage avec les besoins exprimés par le modèle CMAS : notions de priorité et de consommation des événements par exemple. A partir de ce modèle enrichi nous re-concevrons des exemples significatifs d'agents de St Etienne. Puis nous prospectorons des applications vraiment industrielles et appliquerons notre modèle.

Cette expérimentation va nous donner des indications précieuses pour faire évoluer notre idée de modèle de composant idéal. Nous engagerons alors à plus long terme la définition et la mise en œuvre coordonnée de ce modèle idéal qui pourrait être écrit directement au-dessus de Java.

6 Perspectives

La prochaine réunion (le 13 janvier 2004) nous servira à appréhender la définition de l'ajout de fonctionnalité dans Eclipse. Nous prévoyons également une analyse plus fine et plus technique du modèle Fractal par Pierre-Charles David de l'EMN. Nous continuerons l'analyse du modèle et des besoins des agents par rapport au modèle de composant.

La réunion suivante aura lieu de façon couplée avec LMO. Une présentation du langage ArchJava (Sebastian Pavel, EMN) est prévue. Nous fixerons également les spécifications des besoins du modèle à ce moment là. Le prototype du modèle à composant est planifié pour la rentrée de septembre, nous devons disposer d'un modèle à composant intégrant les besoins essentiels pour les agents.

Une journée « Des agents et des composants » sera organisée à l'occasion de la conférence nationale sur les agents (JFSMA) qui a lieu à Paris en novembre 2004. Nous pensons avoir un ou deux invités pertinents sur le sujet (Jean-Pierre Briot du LIP6 ou Martin Wirsing de la TUM). A l'issue de cette journée et si l'opportunité semble crédible un appel pour un numéro spécial d'une revue sur le thème pourrait être lancé.

Pour l'année suivantes les grandes lignes sont : applications agents industrielles et revue du modèle de composant. A plus long terme ce premier travail nous permet de poser les bases d'une collaboration plus durable, la suite pouvant être l'implémentation propre du modèle de composant idéal. Ce modèle intégrant des possibilités de programmation par aspects, la gestion d'exception, etc. Des encadrements communs de thèses seront envisagés dans ce cadre.

Références

- [1] Mohamad K. Allouche. *Une société d'agents temporels pour la supervision de systèmes industriels*. Thèse de doctorat, Ecole Nationale Supérieure Mines de Saint-Etienne & Université Jean Monnet Saint-Etienne, 1998.
- [2] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with JADE. In C. Castelfranchi and Y. Lésperance, editors, *Intelligent Agents VII : Agent Theories, Architectures, and Languages (ATAL 00)*, volume 1986 of *LNAI*, pages 89–103, Boston, MA, USA, 2001. Springer-Verlag.

- [3] L. Boloni and D.C. Marinescu. A component agent model – from theory to implementation. In *2nd International Symposium on Agent Systems and Applications and 4th International Symposium on Mobile Agents (ASA/MA 2000)*, volume 1882, pages 99–112. Springer-Verlag, 1999.
- [4] N. Bouraqadi. Metaclasstalk - a testbed for exploring programming paradigms. Talk given at the European Smalltalk Users Group (ESUG) 10th Smalltalk Conference, Douai - France, August 2002.
- [5] N. Bouraqadi. Efficient support for mixin-based inheritance using metaclasses. In *Workshop on Reflectively Extensible Programming Languages and Systems at The International Conference on Generative Programming and Component Engineering (GPCE'03)*, Erfurt, Germany, September 2003.
- [6] N. Bouraqadi. Simplification de la production de logiciel par la programmation par aspects. Technical Report 2003-3-1, Ecole des Mines de Douai, March 2003.
- [7] N. Bouraqadi. Safe metaclass composition using mixin-based inheritance. *Journal of Computer Languages*, 2004. (to appear).
- [8] N. Bouraqadi and T. Ledoux. Le point sur la programmation par aspects. *Technique et Science Informatique*, 20(4) :505–528, 2001.
- [9] N. Bouraqadi and T. Ledoux. *Aspect-Oriented Software Development*, chapter 11 – Supporting AOP using Reflection. Addison-Wesley, 2004. (to appear).
- [10] T. Bouron. Application des système multi-agents dans les télécommunications. In Jean Pierre Briot and Yves Demazeau, editors, *Principes et Architectures des Systèmes Multi-Agents*, IC2, pages 235–266. Hermès, November 2001.
- [11] F. Bousquet and C. Le Page. Systèmes multi-agents et eco-systèmes. In Jean Pierre Briot and Yves Demazeau, editors, *Principes et Architectures des Systèmes Multi-Agents*, IC2, pages 235–266. Hermès, November 2001.
- [12] M. Bratu, J.M. Andreoli, O. Boissier, and S. Castellani. A software infrastructure for negotiation within inter-organizational alliances. In *AMEC IV*, volume 2531, pages 161–179. Springer-Verlag, 2002.
- [13] F. Brazier, B. D. Keplicz, N. R. Jennings, and J. Treur. Formal specification of multi-agent systems : a real-world case. In V. Lesser, editor, *1st International Conference on Multi-Agent Systems (ICMAS 95)*, pages 25–32. AAAI Press, 1995.
- [14] X. Briffault and S. Ducasse. *Squeak*. Eyrolles, 2001.
- [15] C. Carabelea and P. Beaune. Engineering a protocol server using strategy agents. In *Multi-Agent Systems and Applications III : 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *LNAI*, pages 413–422, Prague, Czech Republic, 2003. Springer.
- [16] Thibault Carron. *Des Systèmes Multi-Agents Temporels pour des Systèmes Industriels Dynamiques*. Thèse de doctorat, Ecole Nationale Supérieure Mines de Saint-Etienne & Université Jean Monnet Saint-Etienne, 2001.
- [17] D.D. Corkill and V.R. Lesser. The distributed vehicle monitoring testbed : a tool for investigating distributed problem solving network. *Artificial Intelligence Magazine*, 4(3) :15–33, 1983.
- [18] Y. Demazeau. Créativité émergente centrée utilisateur. In *Journées Francophones Systèmes Multi-Agents*, pages 31–36, 2003.
- [19] F. Dignum and C. Sierra. Agent mediated electronic commerce : Scientific and technological roadmap. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce : The European AgentLink Perspective*, volume 1991 of *LNAI*, pages 1–18. Springer-Verlag, 2001.
- [20] Andres Farias. *Un modèle de composants avec des protocoles explicites*. PhD thesis, École des Mines de Nantes and Université de Nantes, December 2003.

- [21] J. Ferber and O. Gutknecht. Aalaadin : a meta-model for the analysis and the design of organization in multi-agent systems. In *Proceedings of ICMAS'98*, July 1998.
- [22] Adele Goldberg and David Robson. *Smalltalk 80*, volume 1 – The Language and its implementation. Addison-Wesley, 1983.
- [23] M. Guzdial and K. Rose, editors. *Squeak : Open Personal Computing and Multimedia*. Prentice Hall, 2002.
- [24] S. Hadjab. Assemblage de composants logiciels : les composants composites. Master's thesis, Ecole des Mines de Douai - Université de Technologie de Troye, July 2003.
- [25] Mahdi Hannoun. *MOISE : un modèle organisationnel pour les systèmes multi-agents*. Thèse de doctorat, Ecole Nationale Supérieure Mines de Saint-Etienne & Université Jean Monnet Saint-Etienne, December 2002.
- [26] William Harrison and Harol Ossher. Subject-Oriented Programming (a Critique of Pure Objects). In *Proceedings of OOPSLA'93*, pages 411–428, Washington D.C., September 1993.
- [27] J. Hubner, J.S. Sichman, and O. Boissier. Moise + : Towards a structural, functional, and deontic model for the mas organization. In C. Castelfranchi and W.L. Johnson, editors, *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 02)*, pages 501–502. ACM Press, 2002.
- [28] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Allan Kay. Back to the Future. The Story of Squeak, A Practical Smalltalk Written in Itself. In *Proceedings of OOPSLA'97*, pages 318–326, Atlanta, Georgia, October 1997. ACM.
- [29] M. Kersten and G. C. Murphy. Atlas : a case study in building a web-based learning environment using aspect-oriented programming. In L. Northrop, editor, *Proceedings of OOPSLA'99*, pages 340–352, Denver, Colorado, November 1999. ACM Press.
- [30] G. Kiczales, editor. *Proceedings of the 1st international conference on Aspect-Oriented Software Development (AOSD)*, Enschede, The Netherlands, April 2002. ACM Press.
- [31] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. Getting started with aspectj. *Communications of the ACM*, 44(10) :59–65, October 2001.
- [32] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.
- [33] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings of ECOOP'97*, number 1241 in LNCS, pages 220–242, Jyväskylä, Finland, June 1997. Springer-Verlag.
- [34] sit M. A editor. *Proceedings of the 2nd international conference on Aspect-Oriented Software Development (AOSD)*, Boston, Massachusetts, March 2003. ACM Press.
- [35] Pattie Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of OOPSLA'87*, pages 147–155, Orlando, Florida, 1987. ACM.
- [36] E. Mendizabal, S. Vauttier, and C. Urtado. Un autre modèle de relation d'association pour améliorer la réutilisation de composants de modèles UML. Technical Report RR03/G1/014, LGI2P, novembre 2003.
- [37] T. Meurisse and J.P. Briot. Une approche à base de composants pour la conception d'agents. *RSTI Technique et science informatiques, numéro spécial : Réutilisation*, 20(4) :583–602, 2001. Michel Dao and Christophe Dony editors.
- [38] R. Nassrallah. Programmation par aspects et services web. Master's thesis, Ecole des Mines de Douai - Université de Technologie de Troye, July 2003.

- [39] O. Nierstrasz. Regular types for active objects. In O. Nierstrasz and D. Tschritzis, editors, *Object-Oriented Software Composition*, chapter 4, pages 99–121. Prentice Hall, 1995.
- [40] Equipe OBASCO. Obasco research group. Technical report, INRIA/EMN, 2003. <http://www.emn.fr/x-info/obasco>.
- [41] A.S. Rao and M.P. Georgeff. Modelling rational agents within a BDI-architecture. In J.F. Allen, R. Fikes, and E. Sandewall, editors, *3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, 1991.
- [42] P.M. Ricordel. *Programmation Orientée Multi-Agents : Développement et Déploiement de Systèmes Multi-Agents Voyelles*. PhD thesis, Thèse de doctorat, Institut National Polytechnique de Grenoble, 2001.
- [43] R. Robbes. Mise en oeuvre de la programmation par aspects dans le cadre des systèmes multi-agents. Master's thesis, Ecole des Mines de Douai - Université de Caen, September 2003.
- [44] D.J. Robinson. A component based approach to agent specification. Master's thesis, Air Force Institute of Technology, 2000.
- [45] Jean-Claude Royer and Michael Xu. Analysing Mailboxes of Asynchronous Communicating Components. In D. C. Schmidt R. Meersman, Z. Tari and al., editors, *On The Move to Meaningful Internet Systems 2003 : Coopis, DOA, and ODBASE*, volume 2888 of *LNCS*, pages 1421–1438. Springer Verlag, 2003.
- [46] A. Schmidmeier. Using aspectj to eliminate tangling code in eai activities. Practitioner Reports in the 2nd international conference on Aspect-Oriented Software Development (AOSD), March 2003. Boston, Massachusetts.
- [47] D. Sharp. Aosd for avionics software product lines : Experiences and plans. 1st international conference on Aspect-Oriented Software Development (AOSD). Invited Session : Early Industrial Experience With AOSD., April 2002. Enschede, The Netherlands.
- [48] A. Slodzian. *A Componential Methodology for Modeling Multi-Agent Cooperation*. PhD thesis, VUB Artificial Intelligence Laboratory, 1998.
- [49] Brian C. Smith. Reflection and Semantics in Lisp. In *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages, POPL'84*, pages 23–35, Salt Lake City, Utah, USA, January 1984.
- [50] Diego De Sogos. Component generators : Towards adaptable and efficient software components. Technical report, EMOOSE : Vrije Universiteit Brussel and École des Mines de Nantes, August 2003.
- [51] F. Souchon, C. Dony, C. Urtado, and S. Vauttier. A proposition for exception handling in multi-agent systems. In *Proceedings of the 2nd international workshop on Software Engineering for Large-Scale Multi-Agent Systems*, pages 136–143, Portland, Oregon, May 2003.
- [52] F. Souchon, C. Urtado, S. Vauttier, and C. Dony. Exception handling in component-based systems : a first study. In J.L. Knudsen A. Romanovsky, C. Dony and A. Tripathi, editors, *Proceedings of the Exception Handling in Object-Oriented Systems workshop at ECOOP 2003*, pages 84–91, Darmstadt, Germany, July 2003.
- [53] C. Szyperski. *Component Software. Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, 1998.
- [54] P. Tarr, H. Ossher, and S. M. Sutton. Hyper/j : multi-dimensional separation of concerns for java. In J. Magee and M. Young, editors, *Proceedings of the 24th international conference on Software engineering (ICSE)*, pages 689–690, Orlando, Florida, May 2002. ACM Press. Tutorial.
- [55] S. Vauttier, C. Urtado, and E. Mendizabal. La relation d'association dans les approches dirigées par les modèles : besoins et usages pratiques. In *Actes du colloque ALCAA 2003 (Agents logiciels, coopération, apprentissage et activité humaine)*, pages 151–163, Bayonne, France, septembre 2003.

- [56] Laurent Vercouter. *Conception et mise en oeuvre de systèmes multi-agents ouverts et distribués*. PhD thesis, École des Mines de Saint-Étienne and Université Jean Monnet, December 2000.
- [57] T. Wittig. *ARCHON : An Architecture for Multi-Agent Systems*. Ellis Horwood : Chichester, England, 1992.
- [58] Kaiye Xu. Analysis and implementation asynchronous component model. Technical report, EMOOSE : Vrije Universiteit Brussel and École des Mines de Nantes, August 2003.