Frameworks, architectures et composants: revisiter le développement de systèmes multi-agents

Victor Noël*, Jean-Paul Arcangeli*

*Institut de Recherche en Informatique de Toulouse
Université de Toulouse
118, route de Narbonne, 31 062 Toulouse Cedex, France
{victor.noel,jean-paul.arcangeli}@irit.fr

Résumé. Motivés par le développement de systèmes multi-agents, nous explorons dans ce papier la production de *frameworks* agents en utilisant des architectures logicielles à composants. Nous introduisons brièvement la méthode de développement SPEARAF pour répondre à cette problématique. La contribution de cette article réside pour l'essentiel dans la description du modèle de composants SPEAD qui introduit un type de composants spécifique, le composant *transverse*, qui permet de connecter des composants à une *infrastructure* à l'aide d'une *fabrique d'agents*.

1 Conception des systèmes multi-agents : de nouveaux paradigmes de développement

La **modélisation** d'un problème et/ou de sa solution par un système multi-agent (SMA) est une manière de produire des logiciels. De tels **systèmes** sont composés d'entités, les **agents**, qui interagissent au sein d'un **environnement**. La particularité de cette modélisation est que l'effort de conception est focalisé sur la définition des **comportements individuels** des agents, pour permettre au système d'exhiber un **comportement global** répondant aux exigences initiales. Ces comportements sont basés sur des **interactions** plus ou moins complexes entre les agents, le plus souvent locales, c'est-à-dire ne s'appuyant pas sur des informations globales au système, ce qui fait des SMA une approche particulièrement adaptée aux systèmes distribués, décentralisés ou acentrés. La modélisation par SMA est principalement utilisée dans le domaine de l'intelligence artificielle (résolution de problèmes, systèmes adaptatifs, etc.), de la simulation (en particulier sociale), robotique (collectifs de robots) ou de l'intelligence ambiante (auto-composition de services). [1]

Dans ce contexte, beaucoup de travaux se sont intéressés à l'aide à la **conception** de ces systèmes et un certain nombre de **méthodes de développement** en ont résulté (Henderson-Sellers et Giorgini, 2005). Ces méthodes aident à identifier les agents du SMA réalisé, les moyens d'interaction, directs ou indirects, qui leur sont utiles (échange de messages, environnements partagés, organisations sociales, etc.) et les éléments de leur environnement. Ensuite, en se

1. N'étant pas le propos ici, nous ne justifierons pas l'utilisation des SMA dans ces cadres.

basant sur différentes **approches théoriques**, ces méthodes proposent des guides pour définir le comportement des agents exploitant ces mécanismes d'interaction : toute la difficulté dans cette étape est de déterminer les comportements individuels adéquats qui permettront au système global d'avoir le comportement global escompté.

À partir du résultat de cette phase de conception, le **développeur du SMA** n'a « plus qu'à » **implémenter** le système, ses agents, ses interactions, tout l'outillage associé (interfaces graphiques, *scheduler*…) et à l'intégrer à des systèmes informatiques existants le cas échéant. Heureusement, la plupart des méthodes fournissent des **outils** pour faciliter ce travail en fonction des artefacts produits lors du processus de conception. En revanche, elles se cantonnent le plus souvent à accompagner le développement pour le processus suivi dans la méthode et ne prennent pas en compte les **spécificités** du domaine auxquelles elles sont appliquées.

En effet, chaque méthode, chaque théorie et même chaque SMA a son **propre type d'agents** avec ses propres mécanismes d'interactions, ses propres manières de les exploiter et ce de manière adaptée à un domaine particulier. C'est en ce sens que la **programmation agent** se différencie de paradigmes de programmation plus classiques (objet par exemple) : chaque type d'agents peut être vu comme une machine abstraite avec ses entrées, ses sorties, sa sémantique et sa dynamique. Alors, programmer le comportement d'un agent c'est programmer la machine abstraite. Chaque type d'agents apporte donc son propre **paradigme de programmation** avec ses propres primitives. Programmer en objet, c'est traduire tous les concepts manipulés, et en particulier les interactions, en terme de définition et d'appel de méthode. Différemment, les SMA mettent en avant une diversité de modes d'interaction, entre autres envoi de messages, communication de groupe, interaction indirecte par stigmergie [2], organisation explicite avec sa propre dynamique, etc. Et donc, implémenter un SMA demande un effort supplémentaire pour à la fois mettre en oeuvre ce nouveau paradigme de conception et de programmation, puis pour l'utiliser pour développer son SMA. Certaines **plateformes** à agents tentent de répondre à cette problématique en proposant des types plus ou moins générique d'agents, mais souvent en visant une méthode, une théorie particulière ou un domaine d'application.

À l'opposé, et s'inspirant des approches utilisant les architectures logicielles et les composants logiciels, nous nous sommes intéressés à des moyens de produire, « à la carte », des plateformes à agents qui seraient **adaptées** à un domaine donné [3] et à ses développeurs. Pour cela, nous proposons de fabriquer des **architectures à composants** pour réaliser des ***frameworks*** agents dédiés qui fourniront ces plateformes à agents voulues, ainsi qu'un cadre pour programmer les agents exécutés dedans. Définir de telles architectures à composants revient à mettre en oeuvre les machines abstraites capables d'exécuter le comportement programmé dans le paradigme correspondant au type d'agents réalisé. Notre **apport** à cette problématique est double : en terme de **méthode** (modèles et processus) pour aider à aborder ce problème, et en terme de **modèle de composant** pour appliquer cette approche.

Ce travail s'intègre dans une étude plus générale des apports mutuels entre les composants logiciels et les agents logiciels, ces deux concepts étant communément vus comme deux extensions de l'objet.

Dans la suite de cet article, nous proposons de présenter le modèle de composant que nous avons défini en mettant en avant ses particularités issues du contexte des SMA. Pour motiver ce

2. Dépôt et captage de « phéromones » dans un environnement situé. Souvent utilisé en intelligence artificielle.
3. Nous utiliserons indistinctement les termes « domaine » et « application » lorsque nous parlerons d'artefacts de développement dédiés, adaptés ou spécialisés.

travail, nous dépeindrons dans la section 2 la méthode que nous avons proposé et les problèmes qui nous ont amenés à la création d'un modèle de composant. Puis, nous décrirons le modèle dans la section 3. Enfin nous reviendrons brièvement sur nos objectifs de départ dans la section 4 avant de faire un rapide état de l'art et de conclure.

2 SPEARAF pour concevoir des *frameworks* agents

Un système multi-agent est constitué d'**entités** passives (objets, bases de données...) ou actives (agents, objets actifs...) qui interagissent au sein d'un **environnement** (plan, noeuds d'un réseau, réseaux, structure organisationnelle...) en utilisant des **mécanismes d'action et de perception** (envoi de messages, écriture et lecture sur tableau noir, accès à une caméra, à des roues, mobilité réseau...). L'environnement fait éventuellement office de médium d'interaction (organisation, tableau noir, gestionnaire d'évènements...). Un agent a une **dynamique** (cognitive, réactive...) paramétrée par un **comportement** (connaissances, objectifs, paramètres...) qui le caractérise en tant que individu. La dynamique et le comportement nécessitent des **mécanismes opératoires** (protocoles d'interaction, aptitudes computationelles, raisonnement...) exploitant éventuellement les mécanismes d'action et de perception.

Comme expliqué précédemment, nous avons proposé une méthode, nommée SPEARAF (Species to Engineer Architectures for Agent Frameworks) (Noël et al., 2010) qui fournit des modèles et un processus, mais qui n'entre pas en compétition avec les autres méthodes pour la conception de SMA existantes qui se focalisent sur la fonctionnalité du système. En effet, SPEARAF a pour vocation d'aider à la réalisation de *framework* dédiés à une application en s'appuyant sur l'ingénierie d'architectures à composants.

De tels *frameworks* fournissent ce que nous nommons des « **espèces d'agents** » et des « **écosystèmes** ». Une espèce est un ensemble d'agents avec des caractéristiques structurelles communes. Un écosystème est l'environnement (d'exécution mais aussi applicatif) dans lequel les agents d'une ou de différentes espèces peuvent exister. Notons que les notions d'espèce et d'écosystème différent des notions d'agent et d'environnement utilisées dans les SMA : en effet, ces premières recouvrent autant l'aspect fonctionnel et domaine du SMA à produire (types d'agents, interactions, organisation...), que l'aspect opérationnel nécessaire à la réalisation pratique du SMA (moteur de règles, distribution, cadencement, visualisation...). Ces notions enrichissent celles définies par les méthodes de développement SMA et nous permettent de guider la construction du *framework*. Par exemple on peut définir et implémenter une espèce de robots qui interagissent par messages radios et se déplacent dans un monde virtuel en 2 dimensions, cadencés au tour par tour et visualisables à travers une interface graphique. Ou encore une espèce d'acteurs (au sens de Hewitt et Agha) mobiles dont les membres interagissent par messages sur les noeuds distribués d'un réseau. L'idée ici est de fournir des types d'agent spécifiques qui sont adaptés aux besoins fonctionnels : les développeurs de l'application peuvent donc s'appuyer sur les espèces autant pour concevoir que pour implémenter le SMA **en exprimant ce que les agent « font »** , et ils n'ont donc pas à s'inquiéter de préoccupations opératoires, c'est-à-dire **« comment » les agents font** ce qu'il font ou sont exécutés. Ils peuvent donc se concentrer sur le comportement fonctionnel des agents de leurs applications.

Définir une espèce d'agents signifie à la fois : a) identifier les mécanismes dont les agents de l'espèce ont besoin pour interagir avec d'autres agents et l'écosystème dans lequel ils existent ; b) définir la dynamique des agents de cette espèce (*i.e.* quand et comment les perceptions sont

traitées et les décisions prises) et les mécanismes internes nécessaires à sa mise en oeuvre ; c) définir la dynamique de l'écosystème, en particulier ce qui permet aux agents d'interagir mais aussi d'être créés, ainsi que les mécanismes internes nécessaires à sa mise en oeuvre ; et d) définir le langage (*i.e.* l'abstraction, les primitives...) qui permettra de programmer le comportement des agents de l'espèce. Tout cela revient à définir, « à la carte », un paradigme de conception et de programmation dédié à l'application (au système) à produire.

SPEARAF propose un processus en deux étapes respectivement correspondant à la fabrication d'in *framework* par le **développeur du *framework*** puis à l'utilisation de ce *framework* par l'**utilisateur du *framework*** (programmeur du SMA). Lors de la programmation du SMA, les *hotspots* du *framework* produit sont instanciés par l'utilisateur du *framework* pour implémenter les comportements des agents. Pour réaliser ces *frameworks*, nous nous appuyons sur des architectures à composants. L'idée est de fabriquer pour les agents des architectures dont certains des composants restent abstraits (*i.e.* n'ont pas d'implémentation). Ainsi, les *hotspots* du *framework* basé sur ces architectures seront ces composants, dits de niveau **applicatif**, destinés à être implémentés par les utilisateurs du *framework*. Inversement, les développeurs du *framework* fourniront des implémentations pour le reste des composants, dits de niveau **opératoire**, qui formeront les *frozenspots* du *framework*.

À partir du concept d'espèce, le passage à la fabrication de ces architectures peut se faire plus ou moins systématiquement. Après avoir présenté notre proposition en terme d'architecture dans la section suivante, nous donnerons davantage de détails sur ce point dans la section 4.

3 SPEAD pour décrire et implémenter des architectures agents

Nous avons présenté dans la section précédente une méthode supportant la création de *frameworks* pour les SMA. Dans cette section, nous proposons SPEAD (Species-based Architectural Design), un modèle de composant permettant de réaliser les architectures de ces *frameworks*. Nous nous focalisons ici sur le modèle de composants et les architectures que l'on peut décrire au moyen de ce modèle, et non pas sur les *frameworks* produits.

Les concepts présents dans SPEAD se découpent en 2 niveaux : infrastructure et composant. Notre objectif est de permettre de facilement décrire des composants, de les implémenter et de les utiliser et réutiliser. En particulier, nous avons fait attention à ce que l'articulation entre la description et l'implémentation des composants soit la plus flexible possible en évitant toute répétition et vérification manuelle de la cohérence. Pour cela nous exploitons la génération de code et le typage statique fort du langage cible (ici JAVA ou SCALA).

Nous présenterons d'abord le niveau composant, son modèle ainsi que la façon d'implémenter les artefacts décrits. Puis nous introduirons le niveau infrastructure en plus des composants en motivant sa nécessité.

L'intérêt des composants n'est plus à démontrer en génie logiciel, ce qui nous motive ici est d'abord la réutilisation de mécanismes qui sont récurrents dans les SMA et les agents. De plus, l'utilisation de composants à faible grain et leur capacité d'assemblage nous permet de répondre à notre objectif de fabrication de *frameworks* dédiés à un domaine. Enfin, nous avons exploité la notion de services requis pour définir les abstractions accessibles pour implémenter les *hotspots* des *frameworks* produits.

Pour motiver notre proposition et l'illustrer tout au long de cet article, nous prenons un exemple classique de SMA : une population de fourmis artificielles. Les agents fourmis se dé-

placent dans un monde en 2 dimensions et interagissent par stigmergie : elles déposent des phéromones à leur position courante et peuvent sentir leur présence autour d'elles. La phéromone s'évapore avec le temps. Cette métaphore est souvent utilisée pour résoudre un problème qui se résume à trouver un plus court chemin dans un espace : en effet, avec le comportement adéquat, les fourmis sont capables de trouver un chemin optimal d'un point à un autre en exploitant la stigmergie. Pour nous, cet exemple présente des particularités en terme de SMA : par exemple, les agents vivent dans un environnement et interagissent avec celui-ci (déplacement, dépôt et captage de phéromone), la phéromone a sa propre dynamique d'évaporation et les fourmis ont un comportement défini en terme de perception et d'action. Mais il présente aussi des particularités opérationnelles : par exemple, le système doit être cadencé de manière à ce que les agents aient tous autant de temps d'action, mais aussi pour que les phéromones s'évaporent de façon cohérente. De plus on souhaiterait visualiser l'évolution du système et éventuellement contrôler la vitesse de déroulement. Enfin, en terme d'abstraction, nous souhaitons décrire les comportements des agents sous forme de cycle perception-décision-action.

3.1 Composants

L'idée est donc ici de fabriquer des architectures logicielles qui représentent la partie interne de nos agents. Pour la fourmi par exemple, cela se résume à sa dynamique et son comportement, nous pouvons avoir une architecture très simple faite d'un composant comportemental et d'un composant de cycle de vie. Le premier contient le comportement de la fourmi (par rapport à SpEArAF, sera un *hotspot* de notre framework), il devra implémenter une phase de perception (recevra en entrée les perception de l'instant, c'est-à-dire les phéromones aux alentours), une phase de décision et une phase d'action (utilisera les mécanismes d'interaction avec l'environnement) qui seront accessible sous forme de « services » au reste de l'architecture. Le second est responsable d'exécuter le comportement dans l'ordre précis perception-décision-action en lui donnant les bonnes informations au bon moment (par rapport à SpEArAF, sera un *frozenspot* de notre *framework* implémentant la dynamique des agents). Cette architecture fournira un « service » accessible depuis l'extérieur pour être cadencée et aura aussi des requis vers l'extérieur qui seront réalisés par l'infrastructure, abordée dans la section suivante.

On reconnaîtra (figure 1(d)) les concepts courants utilisés dans ce type de modèles : les composants ont une **définition** avec des ports fournis ou requis typés par une interface (au sens JAVA). Le modèle est hiérarchique et les composants sont **composites** ou **primitifs** (*i.e.* avec ou sans sous-composants, les `Val Component`). Dans les composites sont définis des *bindings* et des délégations pour connecter les ports des sous-composants entre eux et vers l'extérieur (figures 1(i) et 1(h)). Notre proposition n'inclut pas la définition de nouveaux connecteurs ni le typage paramétrique des composants mais nous pensons que ces possibilités peuvent être introduites dans le modèle sans changer les spécificités et avantages de notre proposition.

La définition d'un composant correspond à un type pouvant avoir une ou plusieurs **implémentations**. Si le composant est composite, sa définition correspond aussi à une implémentation de son type sous la forme d'une architecture connectant ses sous-composants. Un composant composite peut donc avoir une ou plusieurs **concrétisations**, c'est-à-dire plusieurs choix d'implémentations pour ses sous-composants (choix pouvant être repoussé jusqu'à l'instanciation du composite).

Ces définitions de composants sont transformées en classes JAVA qui vérifient la même cohérence que la définition en exploitant le typage statique fort du langage. Ainsi, implémenter un

composant primitif revient à étendre une classe abstraite et implémenter les ports fournis (*i.e.* les méthodes de chacune des interfaces). L'accès aux ports requis se fait à travers des membres de la classe abstraite qui implémente la connexion du composant à une future architecture.

3.2 Infrastructure d'exécution

Le définition et l'implémentation de l'infrastructure d'exécution des agents est au coeur de notre proposition. L'infrastructure est ce qui permet de créer des agents, de les exécuter et de les faire interagir. La spécificité de notre proposition est *a priori* liée à la réalisation de SMA, et nous n'avons pu trouver de proposition équivalente dans la littérature. Certains concepts présentés ici feront penser à la notion de connecteur : nous tenterons de mettre en avant les points qui différencient notre proposition des connecteurs.

Motivation. Une infrastructure pour des agents fourmis doit contenir une architecture [4] interne permettant de gérer l'état de l'environnement (maintien des phéromones et leur évaporation, déplacement des agents), mais aussi contrôler le cadencement et permettre la visualisation du système. Nous souhaitons décrire cette infrastructure à l'aide de composants pour sa partie interne, mais aussi entre les agents et l'infrastructure. En effet, dans notre exemple, lorsque l'on ajoute une fourmi au système , on a besoin de connecter l'architecture interne de la fourmi (qui réalise sa dynamique et son comportement) à l'architecture de l'infrastructure.

À un moment donné, du point de vue du système complet exécuté ayant un certain nombre d'agents, il existe une connexion entre chaque agent et l'infrastructure, et ceci pour chacun des mécanismes d'interaction. Cette connexion n'est pas effectuée au déploiement du système, mais à l'exécution car des agents peuvent être créés ou détruits au cours de sa vie. Ici par exemple, chaque architecture d'agent fourmi doit être connectée à l'infrastructure pour : être cadencée, se déplacer et déposer/sentir des phéromones. De plus, ce mécanisme doit parfois être déployé à la fois du côté de l'infrastructure et du côté de l'agent : par exemple pour la stigmergie, l'évaporation est propre à l'infrastructure, le dépôt et la perception sont exécutés dans l'infrastructure du point de vue d'un agent en particulier qui se trouve à une position particulière, et le stock de phéromones est propre à chaque agent. Nous noterons en particulier que l'agent et l'infrastructure sont deux éléments du système de nature différente, le problème ici n'est pas de connecter deux agents entre eux comme l'on connecterait deux composants avec un connecteur, mais bien de connecter un agents à l'infrastructure qui lui permettra d'être exécuté (de « vivre »), et éventuellement d'interagir avec d'autres agents.

Nous voulons donc pouvoir décrire et implémenter à la fois les mécanismes qui se trouvent entre l'agent et l'infrastructure, et à la fois décrire et implémenter la manière de connecter ces mécanismes à l'agent. Nous avons donc à gérer le lien agent-infrastructure, mais aussi des liens entre les différents mécanismes d'interaction d'un agent : par exemple, la position d'un agent est utilisée par le dépôt de phéromone de cet agent particulier.

Ainsi, nous proposons de fournir les modèles nécessaires pour définir de telles infrastructures, mais aussi les implémenter de telle sorte que les mécanismes et composants créés soient aptes à être réutilisés, et de telle façon que la création de nouvelles infrastructures soit aisée pour le concepteur du système. Nous identifions les deux verrous orthogonaux à la réalisation

4. Nous utiliserons indistinctement les termes « architecture » et « composant composite » dans la suite.

de telles infrastructures comme étant la définition et implémentation de moyens d'interaction agent-infrastructure, et leur composition avec l'architecture interne de l'agent à la construction.

Pour répondre à la première problématique, nous proposons les composants *transverses*. Pour répondre à la seconde nous proposons les infrastructure et leurs *fabriques d'agents*.

Solution proposée. Le composant transverse (figure 1(e)) permet de décrire un type de liens entre un agent et une infrastructure et ce de façon réutilisable. Son objectif est de définir des mécanismes permettant l'interaction entre les agents et leur environnements, autres agents inclus. Pour cela, le modèle permet premièrement de décrire ce qu'un transverse fournit ou requiert de l'infrastructure, mais aussi ce qu'il fournit et requiert de chacun des agents qui seront connectés à l'infrastructure à travers. Ensuite, il permet, à l'implémentation, de définir la partie du lien qui se trouve du côté de l'infrastructure, la partie du lien qui se trouve du côté de chaque agent et la fabrique qui permettra d'instancier le lien côté agent à chaque fois qu'un agent est créé. L'interaction entre les deux parties du composant transverse est encapsulée dans l'implémentation du composant.

Les infrastructures (figure 1(g)) sont des agrégats de composants transverses éventuellement connectés à un composant composite représentant l'architecture interne de l'infrastructure. Décrire une infrastructure c'est exprimer les *bindings* entre la partie infrastructure des transverses et le composite interne (figure 1(a)). Concrétiser des infrastructures revient à choisir les implémentations de son composite et de ses transverses.

Dans une infrastructure (et donc étant donné un ensemble de composants transverses), nous pouvons définir autant de fabriques que nous avons de façons de connecter une architecture interne d'un agent à l'infrastructure. Décrire une fabrique, c'est exprimer les *bindings* entre les ports fournis et requis par le composant composite réalisant l'architecture interne de l'agent et les ports fournis et requis de chacun des transverses côté agent (figure 1(c)). Mais c'est aussi définir des *bindings* entre les ports fournis et requis des transverses côté agent entre eux (figure 1(b)). Implémenter une fabrique, c'est définir comment est instanciée l'architecture de l'agent, ainsi que le côté agent des transverses de l'infrastructure. En un sens, une fabrique est la composition des fabriques de transverses avec le composant de l'agent.

Pour permettre de créer des agents (éventuellement par d'autres agents), les fabriques sont disponibles à travers des ports fournis par l'infrastructure.

3.3 SPEADL pour décrire les architectures

Pour permettre la description de ces architectures en pratique, nous avons défini un langage de description d'architectures dont un exemple d'utilisation pour le SMA des fourmis est donné figure 1(f). Cette description n'est pas exactement identique à l'exemple développé dans cet article : nous avons ajouté un composant transverse pour représenter la notion d'agent situé.

4 Exploitation

Retour vers SPEARAF. Pour répondre à notre problématique initiale, voici une ébauche de la façon dont on peut faire correspondre les concepts de l'abstraction fournie par SPEARAF aux concepts de SPEAD :

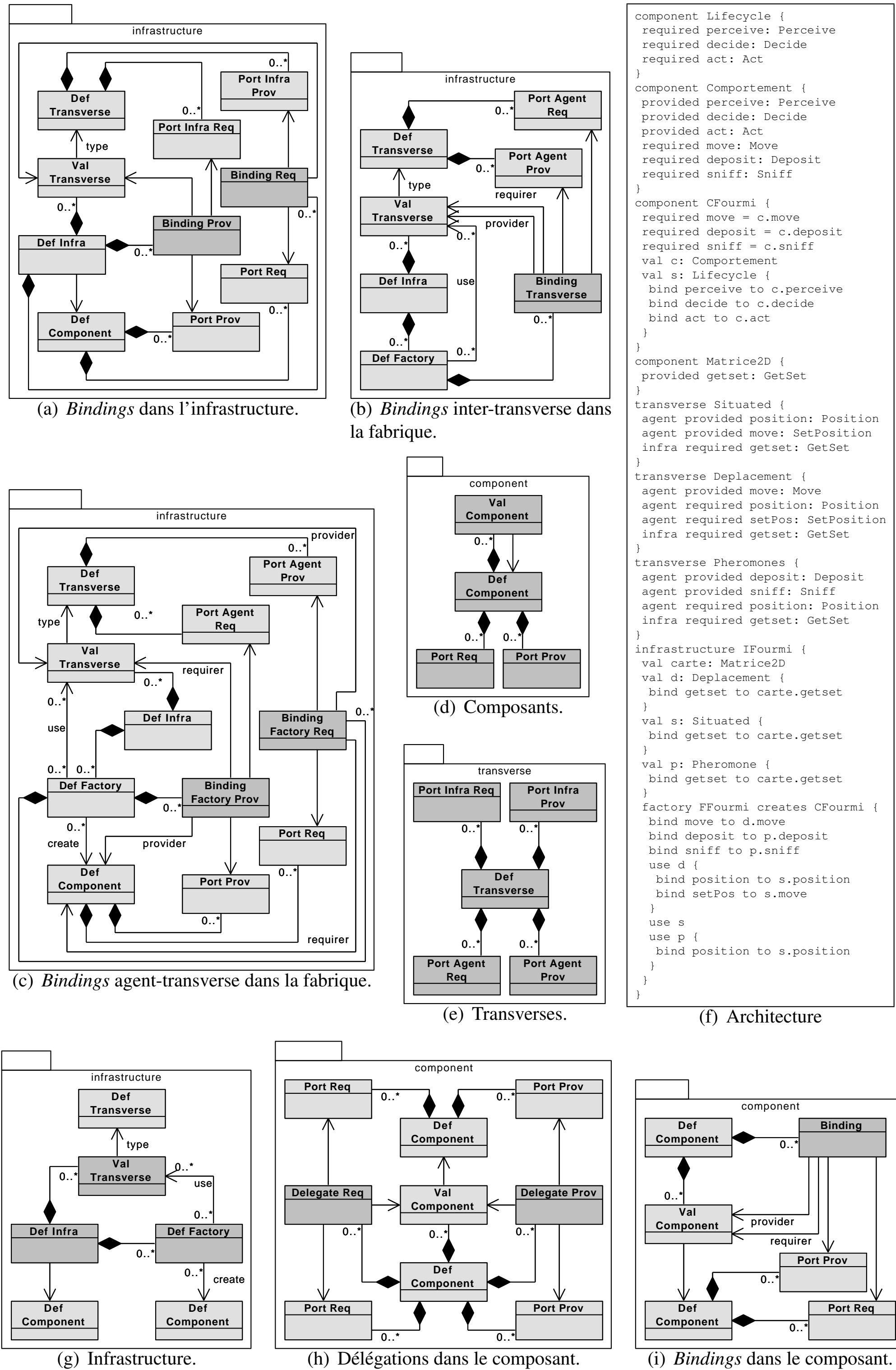

FIG. 1 – *Définition du modèle de composant en UML2 et exemple en* SPEADL.

- l'identification des abstractions nécessaires à l'utilisateur du *framework* ainsi que la définition de la dynamique des agents de l'espèce guidera la conception de l'architecture de l'espèce, des fabriques et le choix des composants abstraits et concrets ;
- l'identification de ce qui compose l'écosystème, que ce soit l'environnement SMA (fonctionnel), d'exécution et de déploiement (opérationnel) ou utilisateur, guidera la conception de l'architecture de l'infrastructure ; et
- l'identification de ce qui permet à l'architecture de l'espèce d'interagir avec l'architecture de l'infrastructure guidera la conception des composant transverses.

Implémentation. Une première version du langage SPEADL a été implémentée sous la forme d'un éditeur textuel et d'un générateur de code, mais sans l'aspect infrastructure, ce qui limite son intérêt. [5] Des premiers prototypes ont été faits pour vérifier la faisabilité de notre approche complète, et une version utilisable est en cours de réalisation. Cette approche est appliquée dans plusieurs projets de recherche, en particulier dans un contexte de robotique distribuée et dans un contexte de simulation sociale.

5 État de l'art

Nous étudions très brièvement quelques travaux comparables au nôtre et mettons en avant ce qui diffère.

Systèmes multi-agents. Plusieurs travaux existent dans le domaine des SMA qui exploitent des composants pour fabriquer des agents (Briot, 2009). Nous retiendrons Generic Agent Model (GAM) (Brazier et al., 1999), MALEVA (Briot et al., 2007) et Magique (Mathieu et al., 2001). Ces travaux portent sur l'utilisation de composants pour fabriquer des architectures d'agents. GAM est un modèle d'agent générique fait de composants avec certains mécanismes d'interaction. MALEVA, de son côté, se focalise sur l'aspect applicatif de l'agent et permet de concevoir le comportement de l'agent avec des composants. Magique permet de faire de même en y ajoutant du dynamisme à l'exécution. Mais tous ces travaux se focalisent sur l'aspect comportemental de l'agent, proposent un type d'agents (*i.e.* une espèce d'agents) figé et ne s'attardent pas sur la partie infrastructure du système.

Composants. Comme dit précédemment, on peut trouver des ressemblances entre les composants transverses et les connecteurs. Par exemple, Medium (Cariou et al., 2002) propose une manière d'implémenter des connecteurs dans un environnement distribué tout en facilitant la réutilisation. On y retrouve la définition d'un type de composants faisant office de moyen d'interaction, qui dans notre approche est réalisé par le composant transverse complété par l'infrastructure. En revanche, nous nous focalisons sur l'intégration d'un composant, l'agent, dans une infrastructure ce qui différencie conceptuellement Medium de notre travail. Bien que moins mature sur le point du composant transverse, notre approche, motivée par les SMA, propose de plus de définir un moyen de connecter le tout à l'exécution en utilisant les fabriques. Cela nous permet de décrire la composition de plusieurs composants transverses à une architecture d'agent.

5. Voir `http://www.irit.fr/MAY`.

6 Conclusion

Dans cette article, motivé par le développement de systèmes multi-agents (SMA), nous avons présenté une méthode pour aider à la fabrication de *frameworks* agents dédiés à un domaine. Pour les réaliser, nous proposons un modèle de composants qui se démarque de l'existant par les deux niveaux de description possibles : composant et infrastructure. En particulier, un type de composants spécifique y est décrit, le composant transverse, qui permet de connecter des composants à l'infrastructure à l'aide d'une fabrique d'agents.

Ce travail nous a permis de confirmer notre intérêt pour les abstractions présentés ici. La suite du travail se focalisera sur les problématiques liées à l'implémentation des composants transverses et des fabriques. De plus, ce travail a été fait dans le cadre d'une recherche plus générale sur le lien entre composants et agents. Entre autres, nous nous intéressons à la question de savoir si la définition d'une espèce d'agents peut s'apparenter à la définition d'un modèle de composants, d'un style d'architecture ou encore d'un conteneur de composants. Un autre axe pertinent est celui de l'utilisation de notre approche pour supporter des lignes de produits logiciels pour les SMA ainsi que l'étude de son intégration avec les méthodes de développement de SMA existantes.

Références

Brazier, F. M. T., C. M. Jonker, et J. Treur (1999). Compositional Design and Reuse of a Generic Agent Model. *Applied Artificial Intelligence Journal 14*, 491–538.

Briot, J.-P. (2009). Composants logiciels et systèmes multi-agents. In A. El Fallah Seghrouchni et J.-P. Briot (Eds.), *Technologies des systèmes multi-agents et applications industrielles*, Chapter 5. Paris, France : Lavoisier.

Briot, J.-P., T. Meurisse, et F. Peschanski (2007). Architectural Design of Component-Based Agents : A Behavior-Based Approach. In R. H. Bordini, M. Dastani, J. Dix, et A. E. Fallah-Seghrouchni (Eds.), *ProMAS 2006*, Volume 4411 of *LNCS (LNAI)*, pp. 71–90. Springer.

Cariou, E., A. Beugnard, et J.-M. Jézéquel (2002). An architecture and a process for implementing distributed collaborations. In *EDOC*, pp. 132–143.

Henderson-Sellers, B. et P. Giorgini (2005). *Agent-Oriented Methodologies*. IGI Global.

Mathieu, P., J.-C. Routier, et Y. Secq (2001). Dynamic Skills Learning : A Support to Agent Evolution. In *AISB'O1, Symposium on Adaptive Agents and Multi-agent Systems*.

Noël, V., J.-P. Arcangeli, et M.-P. Gleizes (2010). Between Design and Implementation of MAS : A Component-Based Two-Step Process. In *EUMAS'10*.

Summary

Motivated by the development of multi-agent systems, we investigate in this paper the production of agent frameworks by using component-based software architectures. We introduce the development method SPEARAF to answer this problem. The main contribution of this article is essentially in the description of the component model SPEAD that introduce a specific type of component, the *transverse* component, that allows to connect components to an *infrastructure* using an *agent factory*.