HAL
archives-ouvertes.fr

Patterns d'Analyse pour l'Ingénierie de Systèmes d'Information à base d'Agents : Une Application au Domaine du Transport

Vincent Couturier, Marc-Philippe Huget, David Telisson

▶ **To cite this version:**

Vincent Couturier, Marc-Philippe Huget, David Telisson. Patterns d'Analyse pour l'Ingénierie de Systèmes d'Information à base d'Agents : Une Application au Domaine du Transport. INFORSID 2011, May 2011, Lille, France. pp.267-282. hal-00622968

HAL Id: hal-00622968
https://hal.archives-ouvertes.fr/hal-00622968

Submitted on 13 Sep 2011

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

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

Patterns d'Analyse pour l'Ingénierie de Systèmes d'Information à base d'Agents : Une Application au Domaine du Transport

Vincent COUTURIER, Marc-Philippe HUGET, David TELISSON

LISTIC – Polytech'Savoie
Université de Savoie
B.P. 80439, F - 74944 Annecy-le-Vieux Cedex
prenom.nom @univ-savoie.fr

RÉSUMÉ. Les Systèmes d'Information de Transport (SIT) peuvent tirer un grand avantage à utiliser des solutions basées sur les agents. En effet, les SIT requièrent une certaine adaptabilité afin de prendre en compte les changements dans les offres et les événements inopinés. Les agents et systèmes multi-agents permettent de répondre à ce besoin d'adaptabilité. Les systèmes d'information à base d'agents comme toute application distribuée, asynchrone et à faible couplage sont difficiles à concevoir et à développer en raison de l'absence de bonnes pratiques d'ingénierie. Cet article décrit une approche fondée sur la réutilisation de patterns aidant à développer de tels systèmes. Les patterns sont des solutions génériques à des problèmes fréquemment rencontrés. Un métamodèle représente et structure les concepts agent. A partir de celui-ci, quatorze patterns d'analyse ont été spécifiés et décrivent les éléments conceptuels nécessaires à la conception de SI à base d'agents. Des patterns de support d'utilisation aident le concepteur à utiliser les patterns précédents lors de l'ingénierie de ces systèmes.

ABSTRACT. Intelligent Transport Information Systems may find benefit of using agent-based solutions. Actually, transport information systems require adaptability to varying changes in offers, and unexpected occurring events. Agents and multiagent systems provide such requirements. Unfortunately, agent-based information systems such as other distributed, asynchronous, loose-coupling applications are difficult to design and implement due to lack of best practices to ease development. This paper describes an approach based on software pattern reuse facilitating engineering of such systems. Patterns are generic solutions to frequently occurring problems. Metamodel represents and structures agent concepts. Fourteen analysis patterns have been specified from this metamodel and describe conceptual entities for the design of an agent-based IS application. Reuse support patterns help designers to reuse former patterns during the information system application engineering.

MOTS-CLÉS: Ingénierie de Systèmes d'Information, Systèmes multi-agent, Patterns, Ingénierie d'applications à base d'agents.

KEYWORDS: Information Systems Engineering, Multiagent systems, Software patterns, Agent-oriented software engineering.

1. Introduction

Le développement des Systèmes d'Information à base d'Agents est une tâche très difficile, car elle repose sur une double complexité : (1) les Agents et, ainsi, les Systèmes Multi-Agents sont des systèmes complexes à modéliser (concepts manipulés) et (2) l'ingénierie de tels systèmes est elle-même complexe.

Résoudre la première complexité consiste généralement à définir l'ensemble des concepts qui sera manipulé et les représenter sous la forme d'un métamodèle. L'ingénierie d'un système à base d'agents répondant à un besoin spécifique consiste donc souvent, lors des phases d'analyse et de conception, à réutiliser ce métamodèle pour représenter les entités spécifiques qui seront par la suite implémentées. Cette réutilisation n'est cependant pas toujours une tâche aisée et, ce, même si une méthodologie (par ex. Gaia (Bernon *et al*., 2004), MaSE (Bernon *et al*., 2004), etc.) y est associée (Deloach *et al*., 2003) (Vafadar *et al*., 2004) (Zambonelli *et al*., 2001). En effet, au moins deux problèmes sont fréquemment rencontrés :

- La documentation associée au métamodèle et/ou la méthodologie de développement est inadaptée. La documentation de certains concepts agent est très souvent implicite alors qu'une documentation explicite favoriserait leur réutilisation. De même, la démarche méthodologique n'est pas toujours suffisamment formalisée; il est ainsi difficile pour un développeur novice de s'y retrouver.

- Un manque de clarté dans les définitions des concepts de base et une absence de guide d'utilisation. Ce manque induit notamment une perte de temps non négligeable lors de la phase d'analyse du système.

Nous proposons, dans cet article, de concevoir et de réutiliser des patterns logiciels pour développer des systèmes d'information complexes tels que ceux reposant sur des agents. Un pattern « décrit un problème qui se manifeste constamment dans notre environnement, et donc décrit le cœur de la solution à ce problème, d'une façon telle que l'on puisse réutiliser cette solution des millions de fois, sans jamais le faire deux fois de la même manière » (Alexander, 1977). Un pattern logiciel (*software pattern*) relève de la même définition et a pour objectif de guider les développeurs en présentant et en explicitant une solution à un problème de construction logicielle dans un contexte spécifique (Beck & Cunningham, 1987). Ainsi, nous proposons de représenter les concepts d'un métamodèle agent sous la forme de patterns d'analyse. Les patterns d'analyse sont des artefacts cognitifs dédiés à la phase d'analyse du processus de développement (Coad, 1996) (Fowler, 1997). Ils définissent la structure d'un agent et représentent les SI à base d'agents à un haut niveau d'abstraction. Notre objectif est de permettre l'ingénierie de systèmes d'information basés sur des agents par réutilisation de ces patterns. Nous proposons également le concept de *Pattern de Support d'Utilisation* aidant le développeur à concevoir et implémenter de tels systèmes en les guidant dans la collection de patterns d'analyse proposée.

Nous présentons en paragraphe 2 le métamodèle agent à partir duquel nous avons dérivé les patterns d'analyse. Le paragraphe 3 présente trois exemples de patterns et le paragraphe 4 précise leur processus de réutilisation. Le paragraphe 5 illustre l'application des patterns d'analyse pour modéliser un Système d'Information de Transport. Enfin, le paragraphe 6 présente les travaux du domaine.

2. Présentation de notre métamodèle

Plusieurs métamodèles ont été proposés dans le domaine de l'ingénierie des systèmes à base d'agents. Ils consistent en général en une extraction des concepts utilisés dans une méthodologie (Bernon *et al*., 2004) (Beydoun *et al*., 2009) et sont ainsi particulièrement limités du fait qu'ils ne concernent que le théorie sous-jacente à cette méthodologie : par exemple, les systèmes complexes ou l'ingénierie des besoins orientés buts. Peu présentent ainsi une grande richesse en concepts agents et seul FAML (Beydoun *et al*., 2009) est suffisamment générique et complet. Notre proposition et celle de FAML partagent la même finalité : fournir un métamodèle générique qui peut être utilisé pour concevoir des applications spécifiques. Notre métamodèle, comme présenté en figures 1 à 4, a ainsi été instancié au domaine de l'ingénierie des systèmes d'information en mettant notamment en exergue les concepts de règles métiers (« business rule ») et de services (Huget *et al*., 2010). Nous n'avons volontairement pas choisi de concevoir nos patterns à partir de FAML, notre métamodèle se montrant plus fin dans la description des concepts et en particulier sur les notions de plan et de protocole.

En fonction de la complexité du SI à concevoir, il pourra être nécessaire de développer différents types d'agents. Le métamodèle proposé permet de définir une grande variété d'agents allant des agents suivant un cycle Perception/Action à ceux suivant un cycle Perception/Décision/Action avec ou sans décision rationnelle.

D'après l'Ingénierie Dirigée par les Modèles (Schmidt, 2006), un métamodèle est la base pour définir les concepts présents dans une application spécifique, le *niveau modèle*. Nous avons adopté la même approche : les concepteurs adaptent les concepts à leur application spécifique, par exemple en ajoutant des propriétés. Un métamodèle n'est cependant pas facilement exploitable par le concepteur, car celui-ci ne sait pas forcément comment faire cette adaptation. Notre travail a consisté à repérer les blocs d'entités du métamodèle liées sémantiquement et pouvant constituer une solution à un problème donné, à associer à chaque bloc un problème et ainsi, in fine, à spécifier un pattern. En outre, chaque pattern permet de guider le concepteur, à l'aide d'exemples d'utilisation, dans son adaptation de la solution afin de représenter un fragment du modèle spécifique de l'application à base d'agents qu'il souhaite développer. Les patterns d'analyse agent permettent ainsi de documenter le métamodèle conçu et de fournir des exemples d'utilisation afin de faciliter la tâche du développeur.

Quatorze patterns d'analyse agent ont été conçus (Cf. figures 1 à 4 pour le positionnement des patterns sur le métamodèle) : *Define System Architecture* (Cf. § 3.2), *Define communication between roles*, *Define protocol*, *Add behavior on message*, *Add behavior*, *Create plan* (Cf. § 3.2), *Define environment* (Cf. § 3.2), *Restrict access to resources*, *Define event*, *Add mental state*, *Expose services*, *Add rules on organisations*, *Define ontology for interoperability* and *Define contracts between agents*.

Le paragraphe 3 présente tout d'abord le formalisme que nous utilisons pour représenter les patterns (§ 3.1) avant de nous attarder sur trois exemples de patterns (*Define System Architecture, Define environment* et *Create plan*). Ces trois patterns (utilisés dans le pattern de support d'utilisation *Base Agent Design*, Cf. § 4) permettent de définir l'architecture de base d'un agent.

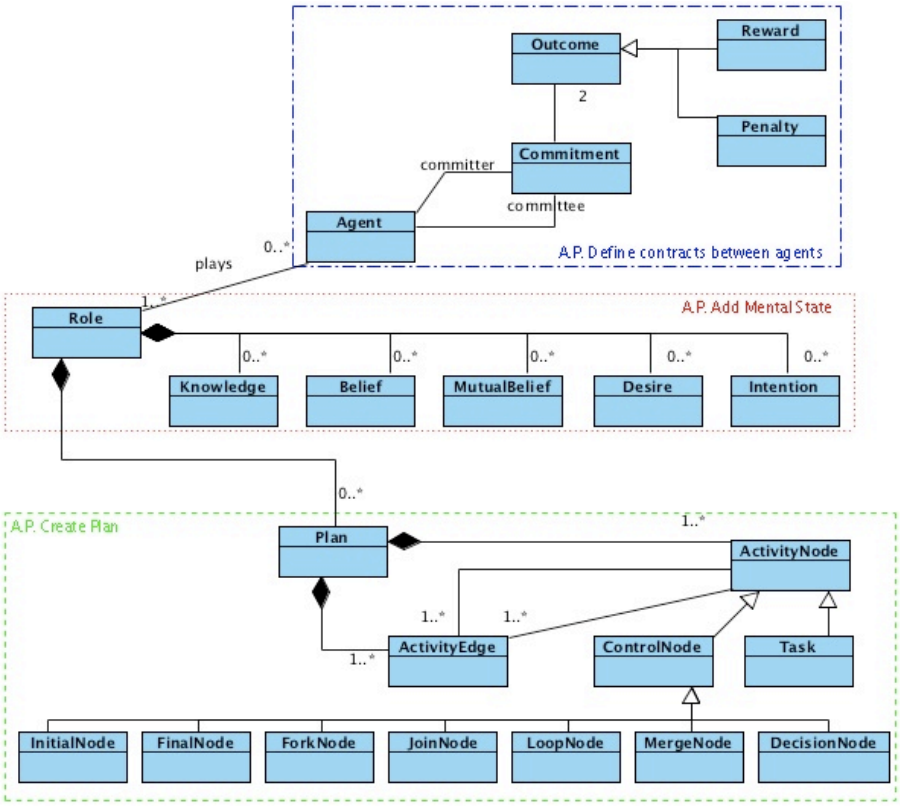


Figure 1. *Vue Agent du métamodèle*

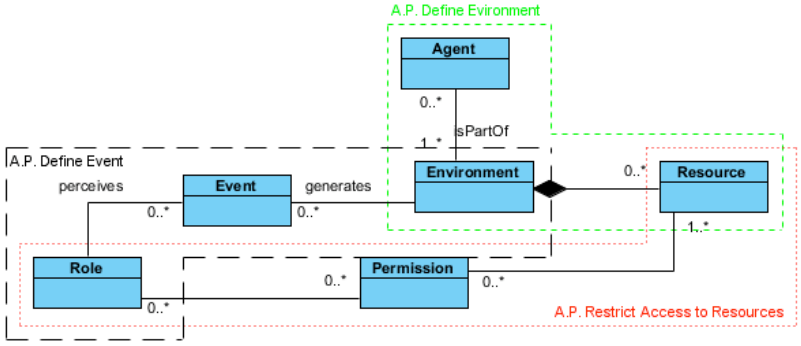


Figure 2. *Vue Environnement du métamodèle*

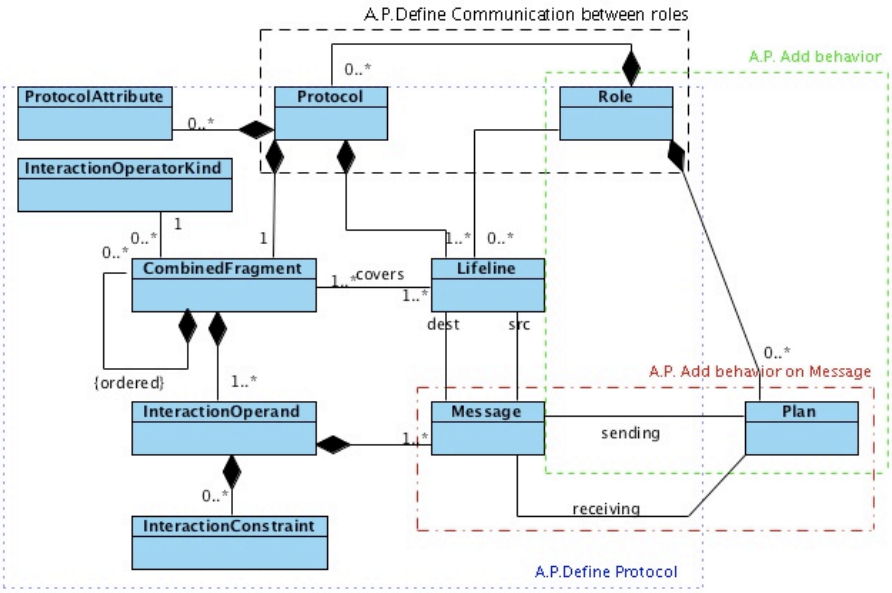


Figure 3. *Vue Interaction du métamodèle*

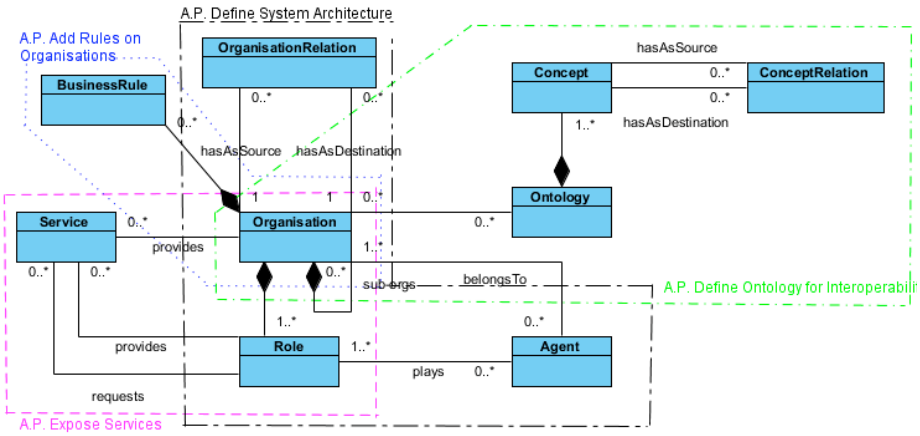


Figure 4. *Vue Organisation du métamodèle*

3. Patterns d'Analyse Agents

3.1. Formalisme des patterns

Le formalisme, utilisé pour spécifier nos patterns, propose des rubriques regroupées en quatre parties.

La partie *Interface*, dédiée à la sélection des patterns, est composée des rubriques *Nom*, *Classification* (détermine le type du pattern : Pattern d'Analyse Agent, Patterns de conception,…), *Contexte* (décrit la pré-condition pour l'application du pattern), *Objectif* (définit le problème adressé par le pattern) et *Applicabilité* (ici, Agent et Systèmes d'information).

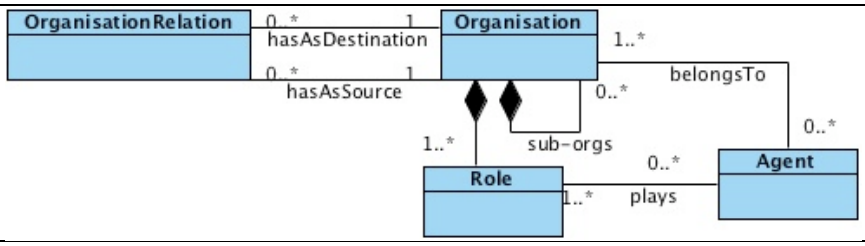
La partie *Solution* exprime la solution d'un pattern en termes de solution modèle ou de solution démarche. Dans le premier cas, elle est constituée des rubriques *Modèle* (un pattern d'analyse propose une solution sous la forme d'un diagramme de classes et/ou de séquences UML), *Participants* (explicitation du rôle de chaque type d'objet du modèle) et *Conséquences* (forces et faiblesses de la réutilisation du pattern pour guider le concepteur dans son choix de pattern). Dans le cadre d'une solution de type démarche (cas, par exemple, des patterns de support d'utilisation, cf. § 4), la rubrique *Modèle* est remplacée par une rubrique *Démarche* proposant une solution sous la forme d'un diagramme d'activités UML.

La partie *Exemples d'utilisation* présente un ou plusieurs exemples d'adaptation – à des cas concrets – de la solution proposée, afin de mieux cerner comment réutiliser le pattern.

Enfin, la partie *Relation* permet d'organiser les relations entre les patterns et donc d'organiser la collection. Elle est constituée des rubriques *Utilise* (Représente la relation : "le pattern X utilise le pattern Y dans sa solution"), *Requiert* ("Le pattern X mentionne le pattern Y dans son contexte, Y sera ainsi appliqué avant X.") et *Raffine* ("un pattern X est un raffinement d'un pattern Y si Y peut résoudre les problèmes posés par X").

REMARQUE — Les patterns proposés en paragraphe 3.2 le sont dans une version succincte. Seules les rubriques les plus importantes à la compréhension du pattern sont présentées. Nous avons ainsi omis de présenter la partie Exemple d'utilisation, décrite en paragraphe 5.

3.2. Exemples de Patterns d'Analyse Agent

Interface
Nom
Define System Architecture
Classification
Pattern d'Analyse Agent
Objectif
L'objectif de ce pattern est de définir l'organisation et les sous-organisations, leurs relations, et les rôles joués par les agents dans ces organisations.
Applicabilité
Agent ^ Systèmes d'information
Solution
Modèle

Participants
Ce pattern décrit la structure organisationnelle d'un système multi-agents (SMA). Un SMA est une *Organisation* éventuellement composée de sous-organisations. Chaque (sous-)organisation est reliée aux autres (sous-)organisations par des *OrganisationRelation*. Le concept *Agent* représente un agent tel qu'il est défini dans la théorie agent, i.e. une entité active, autonome, qui interagit de façon asynchrone avec les autres agents et qui participe à la résolution du problème. Un *Agent* est vu comme l'agrégation d'un ensemble de *Role*. Un agent est identifié de façon unique. Le concept *Role* décrit un rôle que l'agent peut jouer. Un rôle définit un répertoire de comportements joués au sein du système.

Le concept *Agent* est relié au concept *Role* par une association intitulée *plays* et avec les cardinalités suivantes : un *Agent* a 1 à plusieurs rôles, et un rôle peut être joué par plusieurs agents. Le concept *Organisation* définit la structure organisationnelle choisie par les agents. Il peut s'agir d'un groupe, d'une place de marché ou d'une hiérarchie par exemple. Il est possible de définir des hiérarchies (compositions) d'organisations.

Tableau 1. *Pattern d'analyse « Define System Architecture ».*

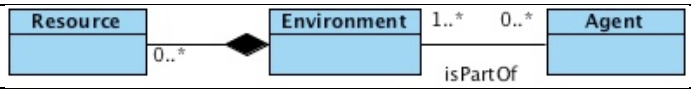
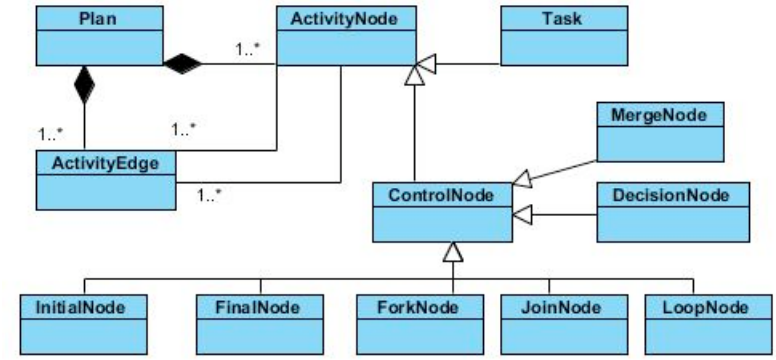
Interface
Nom
Define Environment
Classification
Pattern d'Analyse Agent
Objectif
Ce pattern précise que l'agent est situé dans un environnement qui contient un ensemble de ressources, dans le contexte des systèmes d'information.
Applicabilité
Agent ^ Systèmes d'information
Solution
Modèle

Participants
Le concept *Agent* représente un agent tel qu'il est défini dans la théorie agent, i.e. une entité active, autonome, qui interagit de façon asynchrone avec les autres agents et qui participe à la résolution du problème. Un Agent est vu comme l'agrégation d'un ensemble de rôles. Le concept *Environment* représente le locus dans lequel les ressources et les agents sont immergés. Du point de vue de l'agent, l'environnement représente tout ce qui est extérieur à lui. Nous avons fait le choix de ne représenter au niveau de l'environnement qu'un ensemble de ressources et d'agents. Les agents évoluent dans l'environnement et peuvent percevoir et agir sur les ressources de l'environnement si tant est qu'ils en ont la permission (Cf. Pattern d'Analyse Agent « Restrict access to resources »). Il convient de noter que ce sont les agents qui appartiennent à l'environnement mais que c'est par l'intermédiaire de leur rôle qu'ils accèdent aux ressources de l'environnement. La notion d'environnements simulés cartésiens n'est pas adressée dans ce pattern puisque le pattern porte sur des systèmes d'information. Le concept *Resource* représente des variables partagées entre les agents. Ces ressources peuvent être des documents électroniques tels que des contrats, des propositions, des bases de données pour une entreprise,…

Tableau 2. *Pattern d'analyse « Define Environment ».*

Interface
Nom
Create Plan
Classification
Pattern d'Analyse Agent
Contexte
Ce pattern nécessite l'application préalable du pattern « Add Behavior » et/ou « Add Behavior on Message ».
Objectif
L'objectif de ce pattern est de décrire le comportement sous la forme d'un plan.
Applicabilité
Agent ^ Systèmes d'information
Solution
Modèle

Participants
Le concept *Plan* spécifie le comportement d'un rôle (application du pattern « Add behavior ») ou celui associé à un message (application du pattern « Add behavior on message ») sous la forme d'un ensemble de tâches organisées en flots. Un plan dispose d'un attribut donnant les pré-conditions qui doivent être satisfaites pour pouvoir exécuter le plan et d'un attribut donnant ce qui est vrai après l'exécution du plan (*postconditions*). Un plan est composé d'un ensemble de nœuds (*ActivityNode*) reliés par des transitions (*ActivityEdge*). Les nœuds peuvent être soit des tâches (*Task*), soit des nœuds de contrôle (*ControlNode*) pour modifier le flot. *ControlNode* : représente un nœud de contrôle. Les différents nœuds de contrôle possibles sont : – *InitialNode* qui est unique dans le plan et qui correspond à l'état initial du plan. – *FinalNode* qui indique que le flot se termine avec ce nœud. – *ForkNode* permet de créer plusieurs flots concurrents à partir de ce nœud. – *JoinNode* synchronise plusieurs flots concurrents en un seul flot. – *LoopNode* décrit une boucle avec la condition d'arrêt. – *MergeNode* permet de factoriser le comportement dans le plan pour plusieurs flots. L'ensemble des flots arrivant dans ce nœud effectue par la

suite un comportement unique. Il n'y a pas synchronisation entre les flots. Ceux-ci peuvent arriver à tout moment. – *DecisionNode* sélectionne un flot parmi tous les flots sortants de ce nœud en fonction de conditions Le concept *Task* décrit un comportement atomique que l'agent va réaliser. A cette tâche est associé un ensemble d'attributs : les pré-conditions (*preconditions*) qui conditionnent l'exécution d'une tâche, la description de la tâche (*description*) et les post-conditions (*postconditions*) qui indiquent l'état du monde vrai après exécution de la tâche. Le concept *ActivityEdge* (transition) permet de connecter deux nœuds ensemble. Des conditions sont associées à la transition et doivent être satisfaites pour pouvoir traverser la transition.
Relation
Requiert
Pattern d'Analyse Agent « Add behavior » et/ou « Add behavior on message ».

Tableau 3. *Pattern d'analyse « Create Plan ».*

Nous présentons en paragraphe suivant le processus de réutilisation des patterns d'analyse.

4. Réutilisation des patterns

Comme mentionné en paragraphe 2, nous avons dérivé les patterns d'analyse proposés de notre métamodèle. Même si certains patterns sont reliés entre eux (Cf. partie Relation), cela ne constitue pas une aide suffisante pour décider quels patterns devraient être appliqués et dans quel ordre. Afin de résoudre ce problème, il est nécessaire de fournir une démarche, représentée sous le forme d'un pattern spécifique appelé *Pattern de Support d'Utilisation* (PSU). De tels patterns aident les concepteurs ou développeurs à naviguer dans la collection de patterns et à les réutiliser.

La réutilisation des patterns proposés repose sur l'adaptation. Les concepteurs ne réutilisent ainsi pas directement les patterns d'analyse et le PSU mais adaptent les différents patterns (instanciation) en modifiant leur niveau d'abstraction, conformément à l'application spécifique à développer. En outre, comme mentionné dans le PSU « Service Integration » décrit succinctement ci-dessous, les concepteurs doivent se poser des questions afin de décider quel pattern doit être appliqué. Un autre exemple de PSU, le pattern « Base Agent Design », est présenté en Tableau 4. Il s'applique pour définir l'architecture de base d'un agent.

Interface
Name
Base Agent Design

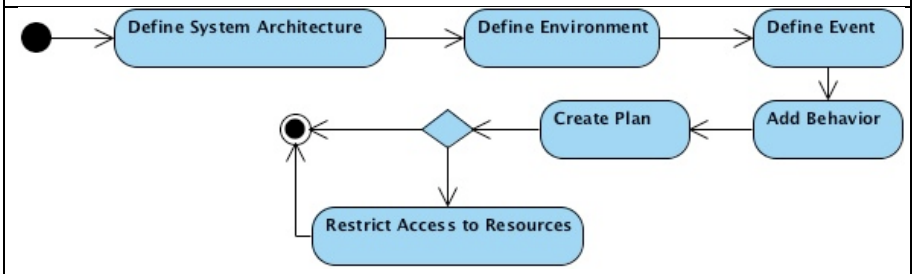
Classification
Pattern de support d'utilisation
Objectif
Ce pattern présente les différents patterns d'analyse agent à mettre en œuvre afin de développer un agent de base. Un agent de base est défini comme un agent jouant des rôles dans des organisations, vivant dans un environnement et réagissant à des événements. Il peut agir sur des ressources - perception et action - s'il dispose des permissions associées.
Applicabilité
Collection de patterns d'analyse agent ^ Agent de base
Solution
Démarche
 Les concepteurs doivent d'abord appliquer le pattern « Define system architecture », puis les patterns « Define environment », « Define event », « Add behavior » et « Create plan ». Après application de ce dernier, il est possible soit de terminer le processus soit d'appliquer le pattern « Restrict access to resources » en fonction de la nécessité de mettre en place des politiques d'accès aux ressources. Cette décision est liée à la place des agents dans l'environnement : les agents accèdent-ils à toutes les ressources ? Doit-on garder des ressources privées ? En fonction des réponses à ces questions, les concepteurs pourront décider d'appliquer ou non le pattern « Restrict access to resources ».

Tableau 4. *Le Pattern de Support d'Utilisation « Base Agent Design».*

D'autres PSU ont été proposés permettant d'adresser des besoins spécifiques. Parmi ceux-ci, nous décrivons brièvement le PSU « Service Integration ». Ce pattern aide les concepteurs à intégrer la notion de service et d'architectures orientées service dans le système d'information à développer. Ce pattern ne se limite pas à lister les patterns d'analyse à appliquer dans un ordre donné mais oblige le concepteur à s'interroger sur la place de l'application à développer au sein de l'ensemble des systèmes d'information de l'organisation :

- Est-il nécessaire d'agentifier les services du système d'information ?
- Doit-on considérer les agents comme des wrappers de services ?
- A-t-on besoin de représenter les comportements des agents sous formes de services ?

- Est-il nécessaire de fournir un accès à des systèmes d'information externes ou à des applications tierces nécessitant ainsi l'interopérabilité et la définition d'ontologies ?

En fonction des réponses à ces questions, le PSU « Service Integration » mettra en exergue un processus spécifique au sein du diagramme d'activités UML proposé dans la partie solution.

5. Etude de cas

Notre étude de cas a pour objectif de fournir une information enrichie aux voyageurs. Cette information voyageur repose en fait sur la collaboration de deux outils spécifiques : (1) un planificateur de trajet prenant en compte des modes de trajets divers (bus, métros, taxis, véhicules personnels, bicyclettes, marche à pieds,…) et (2) un outil d'enrichissement de trajet proposant des points d'intérêt positionnés sur le trajet correspondant aux préférences du voyageur (POI culturels, touristiques,…). Le processus de proposition d'un trajet enrichi est le suivant. Après sélection ou saisie des points de départ et de destination, les systèmes d'information des différents opérateurs de transport (réseaux de bus, de métros, loueurs de bicyclettes,…) coopèrent afin de déterminer les meilleures propositions de trajet correspondant aux préférences (coût, durée, nombre de changements,…) et aux exigences (facilité d'accès, accès handicapés,…) du voyageur. Ensuite, le système sélectionne et détaille les différents trajets proposés par les opérateurs conformément aux exigences et préférences de l'usager. Enfin, les fournisseurs de POI enrichissent les trajets, résultant de la phase précédente, d'informations contextuelles telles que les restaurants, les lieux touristiques (…) correspondant aux préférences de l'utilisateur.

Le système d'information voyageur ainsi présenté repose sur des caractéristiques spécifiques compatibles avec celles des systèmes à base d'agents. Premièrement, la planification de trajet n'est pas réalisée selon une approche classique maitre/esclave. Chaque opérateur est responsable de ses données et est le seul à savoir comment gérer les événements prévus ou inattendus (retards, bouchons, accidents,…). Comme mentionné précédemment, les opérateurs collaborent pour déterminer les trajets optimaux du point d'origine au point de destination. Une deuxième raison repose sur l'ouverture du système : la liste d'opérateurs ou de fournisseurs de points d'intérêt est susceptible d'évoluer au cours de l'exécution. Une troisième et dernière raison est la nécessité pour le système de présenter des caractéristiques d'adaptabilité avancée, afin de prendre en compte notamment la survenance d'événements imprévus. Un SI basé sur des agents nous semble ainsi une nécessité du fait que l'adaptabilité, l'ouverture et la prise en compte du contexte sont des caractéristiques intrinsèques des agents. Nous renvoyons le lecteur à (Wooldridge, 2002), pour de amples détails sur les systèmes à base d'agents et leurs caractéristiques.

Nous nous focalisons dans cet article sur le conception et le développement du Système d'Information de Transport (SIT), en charge de fournir une information voyageur enrichie, par réutilisation de patterns.

Par exemple, l'application du pattern « Define system architecture » nous a permis de mettre en exergue les différents types d'agents conçus (« User Agent », « Travel Planning Agent », « POI Agent » et « Transport Operator Agent ») et leur rôle dans le SIT (Cf. figure 5). Le pattern « Define environment » (Cf. figure 6) nous a permis de définir quels agents et ressources (fichiers plats, fichiers XML, bases de donnés,…) sont situés dans l'environnement.

Les fragments de modèles résultant de l'application des 14 patterns d'analyse ont ensuite été combinés pour représenter le modèle d'analyse du SIT (près d'une centaine de classes).

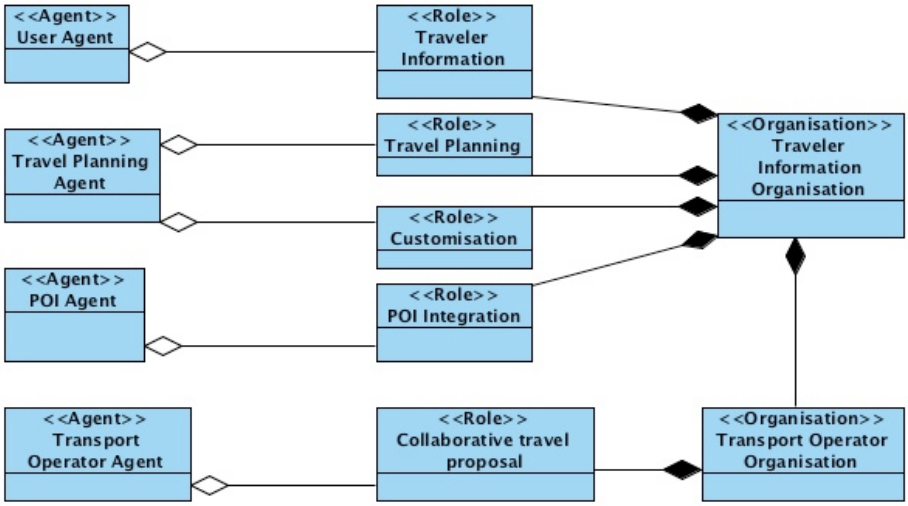


Figure 5. *Instanciation du pattern « Define system architecture »*

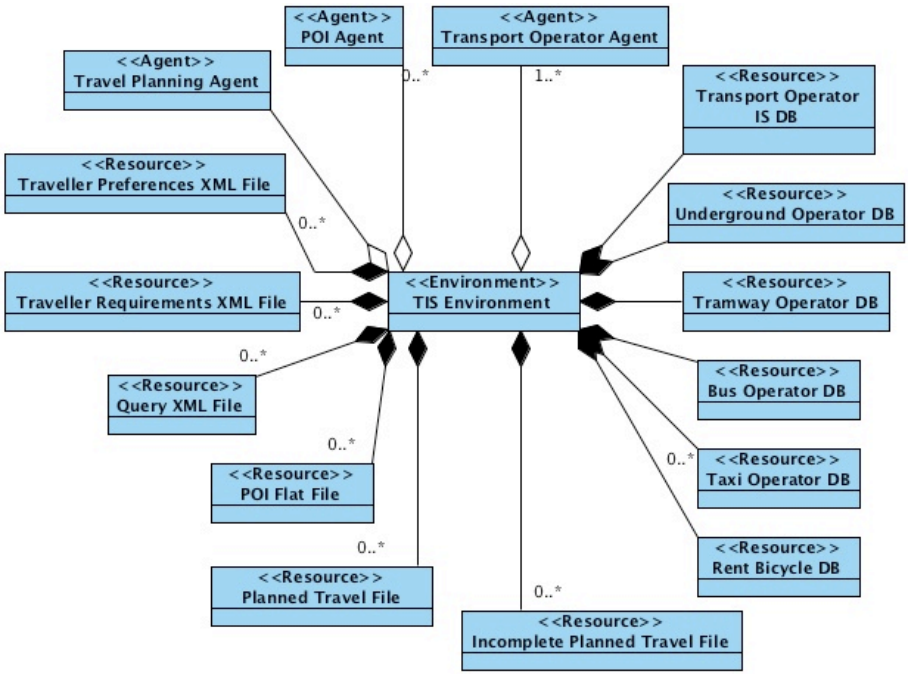


Figure 6. *Instanciation du pattern « Define environment »*

6. Travaux du domaine

Plusieurs patterns ont été proposés dans la littérature pour développer des systèmes à base d'agents. Ces propositions peuvent être classées en deux catégories : (1) propositions spécifiant des patterns sans fournir d'outils ou de processus d'aide à la réutilisation des patterns et (2) des méthodologies reposant sur l'application de patterns.

Les propositions du premier type sont nombreuses et disparates et traitent généralement des problèmes de conception détaillée et d'implémentation des systèmes à base d'agents ((Hung & Pasquale, 1999), (Sauvage, 2003), (Schelfthout *et al.*, 2002),…), mais rarement des problèmes liés à la conception générale (analyse). Ces différentes propositions concernent les agents mobiles (Aridor & Lange, 1998), réactifs ou cognitifs (Meira *et al.*, 2000). Il est difficile de réutiliser ces propositions pour réaliser un système à base d'agents complet : soit la proposition ne concerne qu'un seul type d'agent (notre proposition permet de concevoir n'importe quel type d'agent), soit la collection n'est pas homogène ni

complète et il est difficile pour les concepteurs de déterminer quels patterns doivent être appliqués. En effet, un processus de réutilisation est rarement précisé.

Certaines méthodologies de développement agent sont basées sur l'utilisation de patterns, en particulier Tropos (Do *et al*., 2003) et PASSI (Cossentino *et al*., 2004). Tropos ne propose que des patterns dédiés à la phase de conception détaillée. Ces patterns se focalisent sur les aspects sociaux et intentionnels présents dans les systèmes à base d'agents. Les patterns proposés par PASSI couvrent également la phase d'implémentation, mais il n'est pas aisé de sélectionner les patterns appropriés surtout pour les développeurs novices.

7. Conclusion

Notre objectif est de fournir un ensemble d'outils de développement de systèmes d'information basés sur des agents. Parmi ces outils, nous présentons dans cet article la notion de pattern. Nous avons tout d'abord conçu un métamodèle, aussi exhaustif que possible, en avons extrait des fragments de modèle et les avons intégrés dans des patterns d'analyse. Cette collection cohérente de patterns interdépendants est utilisée pour aider les concepteurs à définir les concepts nécessaires au développement d'une application (phase d'analyse). Ces patterns intègrent des cas d'utilisation basés sur des exemples concrets et sont accompagnés de guides d'aide à leur réutilisation également spécifiés sous la forme de patterns.

Cet article se focalise sur la présentation des patterns d'analyse mais notre travail intègre des patterns couvrant l'ensemble du processus d'ingénierie : patterns d'analyse, patterns d'architecture, patterns de conception et patterns de transformation de modèles (i.e. patterns utilisables lors de la phase d'implémentation). Nous travaillons actuellement à la validation de la complétude de notre proposition en l'appliquant au développement d'une nouvelle application dans le domaine médical. Nous développons également un atelier pour aider les concepteurs à utiliser nos patterns et générer du code pour la plate-forme agent JADE (Bellifemine *et al*., 2007) à partir de nos patterns de conception et de transformation de modèles.

8. Références

Alexander C., Shikawa S., Silverstein M., Jacobson M., Fiksdahl-King I., Angel S., *A pattern language: towns, buildings, construction*, New York, Oxford University Press, 1977.

Aridor Y., Lange D.B., « Agent Design Patterns: Elements of Agent Application Design », *Proceedings of Agents'98*, ACM Press, 1998.

Beck K., Cunningham W., Using pattern languages for object-oriented programs, Technical Report CR-87-43, septembre 1987, Tektronix Inc.

Bellifemine F. L., Caire G., Greenwood D., *Developing Multi-Agent Systems with JADE*, New York, Wiley, 2007.

Bernon C., Cossentino M., Gleizes M.P., Turci P., Zambonelli F., « A study of some multi-agent meta-models », *Agent-Oriented Software Engineering*, Vol. 3382, LNCS, 5th International Workshop AOSE 2004, 2004, Springer, p. 62-77.

Beydoun G., Low G., Henderson-Sellers B. et al., « FAML: A Generic Meta-model for MAS Development », *IEEE Transactions on Software Engineering*, vol. 35, n°6, 2009, p. 841-863.

Coad P., *Object Models – Strategies, Patterns and Applications*, Yourdon Press Computing Series, 1996.

Cossentino M, Sabatucci L., Chella A., « Patterns Reuse in the PASSI Methodology », *Engineering Societies in the Agents World*, Vol. 3071, LNCS, 2004, Springer, p. 294-310.

Deloach S.A., Matson E.T., Yonghua L., « Exploiting agent oriented software engineering in cooperative robotics search and rescue », *International journal of pattern recognition and artificial intelligence*, vol. 17, n°5, 2003, p. 817-835.

Do T.T., Kolp M., Hang Hoang T.T., Pirotte, A., « A Framework for Design Patterns for Tropos », *Proc. of the 17th Brazilian Symposium on Software Engineering*, Brésil, Octobre 2003.

Fowler M., *Analysis Patterns*, Addison-Wesley, 1997.

Huget M-Ph., Couturier V., Telisson D., Proposition d'un métamodèle pour la conception de systèmes multi-agents, rapport de recherche 10/01, 2010, LISTIC, Université de Savoie.

Hung E., Pasquale J., Agent Usage Patterns: Bridging the Gap Between Agent-Based Application and Middleware, technical report CS1999-0638, 1999, Department of Computer Science and Engineering - University of California.

Meira N., Silva I.C., Silva A., « An Agent Pattern for a More Expressive Approach », *Proc. of EuroPLoP 2000*, Allemagne, 2000, p. 331-346.

Sauvage S., Conception de systèmes multi-agents: un thésaurus de motifs orientés agent, Thèse de doctorat, Université de Caen, 2003.

Schelfthout K., Coninx T., Helleboogh A., Steegmans E., Weyns, D.: « Agent Implementation Patterns », *Proc. of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, 2002, p. 119-130.

Schmidt D., « Model-driven Engineering », *IEEE Computer*, vol. 39, n°2, 2006, p. 25-31.

Vafadar S., Barfouroush A.A., Shirazi M., « Towards a More Expressive and Refinable Multiagent System Engineering Methodology », *Agent-Oriented Information Systems*, Vol. 3030, LNCS, 2004, Springer, p. 126-141.

Wooldridge M., *Introduction to Multi-Agent Systems*, Wiley & Sons, 2002.

Zambonelli F., Jennings N., Omicini A., Wooldridge M., « Agent-Oriented Software Engineering for Internet Applications », *Coordination of Internet Agents: Models, Technologies, and Applications*, Chapitre 13, Springer-Verlag, 2001.