Vers une approche d'ingénierie multiagent à base de ligne de produits logiciels

Conference Paper · June 2015

CITATIONS
2

READS
391

4 authors, including:

Anarosa Alves Franco Brandão
Universidade of São Paulo, São Paulo, Brazil
122 PUBLICATIONS **586** CITATIONS
SEE PROFILE

Tewfik Ziadi
Sorbonne Université
82 PUBLICATIONS **1,498** CITATIONS
SEE PROFILE

Zahia Guessoum
Université de Reims Champagne-Ardenne
195 PUBLICATIONS **1,695** CITATIONS
SEE PROFILE

Vers une approche d'ingénierie multiagent à base de ligne de produits logiciels

Anarosa A.F. Brandao[a,b] anarosa.brandao@usp.br
Dounia Boufedji [b,d] dounia.boufedji@lip6.fr
Tewfik Ziadi[b] Tewfik.Ziadi@lip6.fr
Zahia Guessoum[b,c] zahia.guessoum@lip6.fr

[a] *Computing Engineering and Digital Systems Department*
University of Sao Paulo – Brazil

[b] *Sorbonne Université, UPMC Univ Paris 06,*
LIP6, F-75005, Paris, France

[c] *Université Reims Champagne Ardennes*
CReSTIC, F-51000, Reims, France

[d] *Université des Sciences et de la Technologie Houari Boumediene,*
RIIMA, Babezouar,Alger,Algérie

Résumé

Bien que plusieurs méthodes et outils aient été proposés pour l'ingénierie des SMA durant les deux dernières décennies, passer des modèles SMA au code reste une tâche difficile. La majorité de ces méthodes ne parvient pas à proposer une solution pour la réutilisation des implémentations existantes telles que les processus incrémentaux. Notre proposition répond à deux problèmes : combler le fossé entre la modélisation des SMA et l'implémentation d'une part, et fournir une approche incrémentale de développement de SMA en s'appuyant sur les lignes de produits logiciels d'autre part. Cette approche se base sur une description de la variabilité grâce à des modèles de caractéristiques et utilise un framework de ligne de produits logiciels pour la génération des différentes variantes de l'application.

Mots clefs*: SMA, lignes de produits, modèle des caractéristiques, dérivation des produits.*

Abstract

Although several methods and tools to support engineering MAS were proposed in the last decades, it is still a difficult task to go from MAS models to MAS code. Moreover, just a few MAS methods provide guidelines for that and such methods fail in proposing a solution to reuse MAS implementation, as in an incremental process. Our proposal intends to address both issues: filling in the gap between MAS modeling and implementation and providing guidance for incremental development of MAS using a Software Product Line (SPL) approach that goes beyond the variability description through feature models and proposes to generate different variants using existing SPL frameworks.

Keywords*: MAS-PL, product-line, feature model, product generation.*

1. Introduction

Le génie logiciel multiagent a engendré plusieurs méthodes et outils pour l'ingénierie des SMA durant ces deux dernières décennies [4][6][11][18]. Les méthodes proposées couvrent plusieurs étapes du cycle de développement telles que l'ingénierie des besoins, la conception, l'implémentation, etc. Par exemple, ASPECS [18] couvre toutes les étapes, elle est applicable aux SMA holoniques. Cependant, la majorité des méthodes multiagents ne couvre pas l'ensemble des étapes. Elles s'arrêtent souvent à la conception qui élabore des modèles multiagents. Néanmoins, le passage des modèles produits par les méthodes aux SMA opérationnels est un problème compliqué notamment pour les non spécialistes des SMA,

car les outils d'implémentation n'utilisent pas les mêmes modèles et les mêmes concepts que les méta-modèles des méthodes. Par ailleurs, le développement des SMA est souvent incrémental. Il nécessite ainsi, une double expertise à la fois en conception et en implémentation.

Nous pensons que l'écart entre les méthodes et les outils multiagents est dû à une caractéristique du développement des méthodes multiagents. Contrairement à l'approche objets où les méthodes et les langages de modélisation ont émergé de l'implémentation des systèmes à base d'objets, la majorité des méthodes multiagents a été développée sans s'appuyer sur une expertise en implémentation.

Pour combler l'écart entre la conception et l'implémentation et pour faciliter le développement incrémental, la communauté génie logiciel a introduit une nouvelle approche appelée Ligne De Produits logiciels (LdP). La LdP est un paradigme qui prône une vision de modélisation et de développement dans laquelle l'objectif n'est pas l'obtention d'un seul système logiciel à la fois, mais plutôt un ensemble de systèmes logiciels possédant des caractéristiques communes mais aussi qui diffèrent en certains points de variabilité. Ces logiciels peuvent être des variantes d'une famille d'applications similaires. La gestion de la variabilité est la première dimension clé dans l'ingénierie des LdPs. La deuxième dimension concerne la construction, appelée aussi dérivation des membres de la ligne d'un produit. Chaque logiciel variant est obtenu donc par dérivation en instanciant un ou plusieurs points de variabilité.

Différentes approches combinant les SMA et les LdPs proposent des extensions de méthodes SMA existantes afin d'intégrer la notion de variabilité dans les modèles SMA (voir par exemple [7][9][15][16][17]). Cependant, les solutions existantes ne se sont pas focalisées sur l'écart entre les méthodes et les outils d'implémentation multiagents et ne proposent pas des mécanismes permettant d'aller au delà de la description de la variabilité.

Dans cet article, nous proposons une approche pour combler l'écart entre les méthodes et les outils d'implémentation et faciliter le développement incrémental des variantes SMA. Tout d'abord, nous proposons des lignes directrices pour spécifier les modèles de caractéristiques (*features model*) dans le contexte des SMA. Ces caractéristiques pourraient être déduites des méta-modèles, des méthodes existantes ou des applications existantes (code).

Enfin, et pour réduire l'écart entre les modèles et le code, nous réutilisons des environnements de LdPs existants afin de générer des implémentations JADE pour plusieurs variantes de SMA. L'exemple de l'emploi du temps est utilisé pour illustrer notre approche.

Ce papier est organisé comme suit : la section 2 présente un état de l'art sur l'ingénierie des LdPs et discute les travaux existants. La section 3 présente l'exemple de l'emploi du temps. La section 4 décrit notre approche LdP-SMA. La section 5 conclut ce travail et présente quelques perspectives.

2. Contexte et état de l'art

Cette section introduit les LdPs et présente les approches combinant les LdPs et les SMA.

2.1 Lignes De Produits Logiciels (LdP)

Les LdPs sont une transposition des chaînes de production industrielle au monde logiciel. Pour réduire les coûts et le temps de développement, elles visent à produire une famille de systèmes au lieu d'un système unique. Clements et Northrop [8] définissent une LdP comme un ensemble de systèmes logiciels partageant un ensemble commun de fonctionnalités ou caractéristiques qui répondent à des besoins spécifiques d'une partie particulière du marché, ou mission et qui sont développés d'une manière prescrite à partir d'un noyau commun d'éléments.

L'ingénierie des LdPs s'axe sur la capture de points communs et de la variabilité entre plusieurs produits logiciels appartenant au même domaine [8]. Les points communs rassemblent des hypothèses qui sont vraies pour tous les membres de la LdP, tandis que la variabilité regroupe l'ensemble des hypothèses montrant comment les *produits*, membres de la *ligne* de *produits* diffèrent.

L'ingénierie du domaine consiste à développer et construire la LdP en utilisant l'analyse et l'implémentation du domaine.

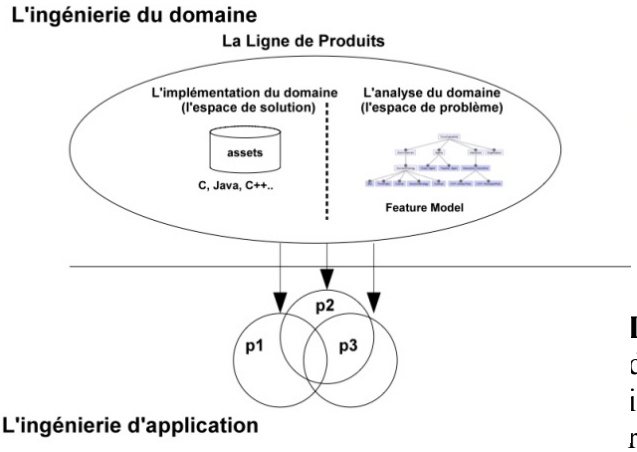

FIG. 1 – Ingénierie de LdP.

Lors de l'analyse du domaine, des points communs du domaine et la variabilité sont identifiés, les membres de la LdP sont ensuite spécifiés en utilisant des modèles de variabilité. Le modèle de caractéristiques (qui sera nommé FM[1] dans ce papier) est le formalisme principal pour la capture de points communs et de variabilité des LdPs [3][14]. Une caractéristique est définie comme un aspect important ou distinctif et apparent, la qualité ou attribut d'un système [14]. Un exemple illustratif d'un modèle de variabilité simple concernant une LdP simple d'une famille de voitures est illustré dans la figure 2. Les éléments communs de cette famille sont spécifiés en utilisant des caractéristiques obligatoires où la variabilité est décrite en utilisant les caractéristiques optionnelles, le *ou*, le *ou exclusif* ; ce qui est également complété par des contraintes.

La notion de *configuration* est utilisée pour représenter un produit d'une LdP. Elle consiste en une sélection de caractéristiques qui sont compatibles avec les contraintes du FM [2]. Dans l'exemple de voitures, une configuration peut contenir le choix des différentes options d'une voiture particulière.

Lors de l'implémentation du domaine, des éléments logiciels sont construits et associés à chaque caractéristique. Un élément logiciel est un artefact, qui est utilisé pour développer les produits logiciels. Les documents de spécification des besoins, les modèles, le code source, etc. sont des exemples de tels éléments.

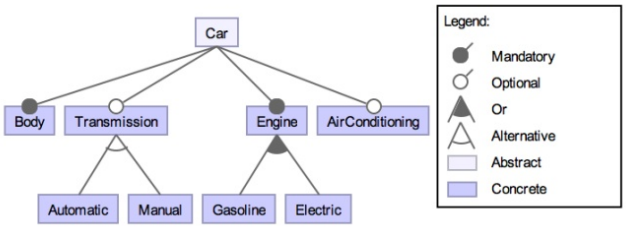

FIG. 2–Modèle de caractéristiques

L'ingénierie d'applications consiste à dériver des produits membres sur la base de la LdP issue de l'ingénierie du domaine. De nombreuses approches existantes sont proposées par la communauté LdP pour la mise en œuvre de la dérivation de produits. Dans cet article nous utilisons FeatureHouse qui offre des services de composition de caractéristiques [1]. Il est générique et peut prendre en charge de nombreux langages de programmation dont Java.

2.2 Approches combinant LdPs et SMA

Cette section décrit et analyse les approches existantes qui combinent les LdPs et les méthodes SMA. Cette combinaison a généré un nouveau paradigme : Les Lignes de Produits des SMA (LdP-SMA) dont le but est d'améliorer l'ingénierie des SMA en combinant les avantages des deux approches.

Dans [17], les auteurs comparent les deux approches et étudient les avantages et les défis de ce nouveau paradigme de l'ingénierie des SMA. Ils considèrent que les deux approches sont basées sur les mêmes concepts dans les premières phases de développement. Par exemple, les deux approches utilisent des modèles dans les phases d'analyse ; les LdPs utilisent des modèles de caractéristiques ; et les méthodes multiagents utilisent les modèles SMA. Ces modèles sont indépendants de la plateforme. Dans la phase d'implémentation, les LdPs s'appuient sur une architecture de base commune et des éléments réutilisables. Cependant, la phase d'implémentation n'est pas souvent prise en compte dans les méthodes AOSE. Plusieurs approches LdP-SMA ont ainsi

[1] **F**eature **M**odel

été proposées dont la majorité utilise Gaia et PASSI.
Dehlinger et Lutz ont introduit Gaia-LdP qui se concentre sur la documentation et la réutilisation des spécifications des besoins pour une LdP-SMA [9]. Leur proposition étend Gaia pour inclure les principes de réutilisation. Gaia-LdP propose deux contributions principales : (i) elle introduit des points de variation dans la conception et le développement des SMA ; (ii) elle propose un modèle de spécification des besoins.
Une autre approche basée sur Gaia, nommée MaCMAS, propose un FM pour documenter les points communs et les variabilités, ce qui permet la description d'une même caractéristique à différents niveaux d'abstraction permettant ainsi de spécifier et de tester les changements à chaque niveau d'abstraction. Cette approche a été proposée par Peña et al. [16] pour faire face à la conception et à la gestion des SMA en évolution en utilisant les LdP-SMA. MaCMAS vise la construction du noyau de l'architecture de base d'une approche LdP-SMA.
Nunes et al. proposent une approche qui s'avère de nos jours être la plus complète car elle recouvre toutes les étapes allant des besoins à l'implémentation [15]. Ils y ont adapté et repris des éléments de la méthode LdP PLUS. Cependant, cette approche utilise un modèle unique pour toute la LdP enrichi avec des annotations pour exprimer toutes les variabilités. Le résultat est en effet un modèle complexe qui est difficile à utiliser par les personnes qui ne sont pas impliquées dans son développement.
La LdP-SMA est une approche prometteuse pour l'ingénierie SMA. En fait, la plupart des développeurs des SMA utilisent une approche incrémentale pour construire des applications SMA et suivre implicitement une approche de ligne de produits. En outre, le paradigme LdP-SMA permet de formaliser la réutilisation dans le développement des SMA. Cependant, la plupart des solutions existantes introduisent des notations complexes qui sont difficiles à comprendre et à utiliser. Par ailleurs, elles ne sont pas adaptées au développement incrémental des SMA car elles sont basées sur des méthodes existantes qui ne considèrent pas la phase d'implémentation. Ainsi, notre approche est différente car nous proposons de partir de l'implémentation d'un SMA en se reposant ainsi sur une approche LdP extractive, où nous étudions les points de variation et définissons les caractéristiques.
Les LdP-SMA sont très proches des méthodes d'ingénierie à base de modèles. Cependant, ces dernières méthodes visent à développer un seul système alors que les LdP-SMA permettent de générer une famille de systèmes. Ils sont en effet mieux appropriés au développement incrémental.

3. Exemple

Pour illustrer notre approche, nous proposons d'utiliser le problème de l'emploi du temps. Dans cette section, nous commençons par présenter une variante simple de cet exemple, puis un ensemble de facteurs de variabilité est introduit.
Description : Le *benchmark* emploi du temps (appelé *TimeTable* ci-après) a été proposé par Bernon et al. [5] afin d'étudier certains problèmes dans le domaine des SMA. Il représente un problème d'allocation de ressources complexe pour lequel la recherche de solutions doit être collective.
Le diagramme de classes UML de la figure 3 résume l'implémentation de la variante simple de l'emploi du temps. Il comprend deux parties: 1) un ensemble de classes qui représentent l'ontologie du domaine et 2) un ensemble de classes représentant les agents et leurs comportements. Cette variante simple du problème de l'emploi du temps ne considère aucune contrainte concernant les ressources. Par exemple, elle considère que les enseignants et les groupes d'étudiants trouvent une solution pour les cours. Les classes sont ensuite affectées à ces cours avec tout le matériel nécessaire (Tableau noir, vidéo projecteur ...) et les contraintes (taille de la pièce ...).
Dans notre modèle, chaque utilisateur est doté d'un agent assistant dont le comportement est guidé par le protocole contrat Net (CNP) :

- L'agent Enseignant (*Teacher_Agent*) a pour objectif d'assurer sa charge d'enseignement (en créneaux) en fixant les créneaux et les cours qu'il doit assurer à toutes les classes (groupes d'étudiants).
- L'agent Classe (*Class_Agent*) représente un groupe d'étudiants auquel un enseignant sera assigné pour un cours spécifique.

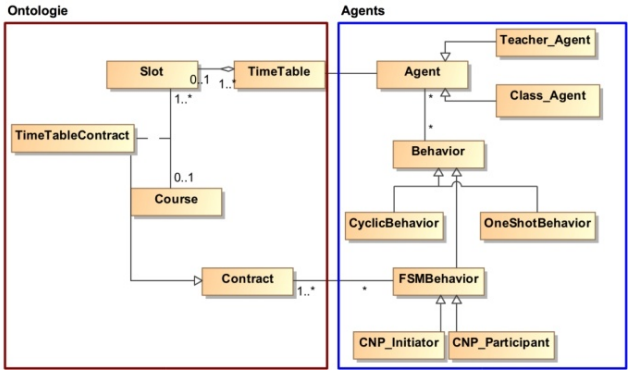

FIG. 3 – diagramme de classes UML pour l'exemple simple

Variabilité : La variante de l'emploi du temps décrite ci-dessus est simple. Toutefois, le problème de l'emploi du temps peut avoir plusieurs variantes de part l'inclusion ou l'exclusion de contraintes qui peut être très complexe [5]. En effet, si on considère le problème associé à ce système et qui consiste à trouver un emploi du temps cohérent en tenant compte des points suivants : (i) une liste de *m* enseignants, (ii) une liste de *n* classes (groupes d'élèves) et, si nécessaire, (iii) une liste des *p* salles. Compte tenu de ces nouvelles considérations, plusieurs variantes peuvent être déclinées. En effet, l'introduction de la variabilité concernant le nombre d'agents entraîne le changement du protocole d'interaction. En plus, la prise en charge des salles génère aussi d'autres variantes en fonction de la façon dont les ressources sont gérées.

Dans la section suivante, nous allons présenter notre approche pour utiliser les LdPs afin d'implémenter les SMA. Elle suit l'environnement général des LdPs présenté dans la section précédente. Nous commençons par montrer comment spécifier un FM. Nous montrons ensuite comment les différentes variantes du système de l'emploi du temps peuvent être obtenues en utilisant une approche de composition de LdP.

4. Id-MASPL[2]

Notre nouvelle approche LdP-SMA, nommée Id-MASPL, vise à traiter deux questions:1) le développement incrémental d'une famille de SMA du même domaine, et 2) la réduction de l'écart existant entre la modélisation et l'implémentation des SMA.

Nous proposons, en premier lieu, un canevas de FM des SMA. Nous montrons par la suite, comment raffiner ce FM pour prendre en considération les nouveaux facteurs de variabilité. Enfin, nous illustrons les questions d'implémentation concernant les artefacts logiciels et dérivation de variantes.

4.1 Vers un modèle pour les caractéristiques des SMA

Dans cette section, nous présentons une approche de LdP pour l'implémentation des SMA ou la réutilisation de solutions SMA existantes pour les raffiner ou les étendre.

Etant donnée une représentation abstraite d'un SMA, qui pourrait être le produit d'une méthode multiagent, notre idée est de construire un FM basé sur cette représentation abstraite et sur le paradigme Voyelles (Agent, Environnement, Interaction, Organisation) proposé par Yves Demazeau [10].

Notre FM est construit à partir du modèle SMA élaboré pour l'implémentation de la première variante de l'exemple. Il possède deux niveaux : un niveau abstrait, et un niveau spécifique au domaine. Au niveau abstrait nous considérons les principales abstractions qui font partie d'un SMA, selon le paradigme Voyelles, pour organiser les caractéristiques abstraites. Au niveau spécifique du domaine, en dessous de chaque abstraction correspondant à une voyelle, nous retrouvons les caractéristiques liées à l'Environnement, aux Agents, à l'Interaction et à l'Organisation.

Dans notre exemple, la description abstraite du système présentée dans le diagramme UML doit

[2] **I**ncremental **d**evelopment of **M**ulti-**A**gent **S**ystems with Software **P**roduct **L**ine

être mise en correspondance avec un FM composé des caractéristiques spécifiques du domaine

Caractéristiques de l'environnement : L'environnement des agents représente le contexte dans lequel les agents sont situés. Chacune des classes de l'ontologie du domaine (*Course*, *Slot* et *TimeTable*) est en effet associée à une caractéristique.
La classe *Contract* représente la stratégie de décision adoptée par les rôles d'interaction pour accepter et / ou rejeter les propositions reçues. Par conséquent, une caractéristique *Decision-Strategy* lui est associée.

Caractéristiques des Agents : La mise en correspondance est directe car les agents sont clairement identifiés dans le diagramme UML et chaque classe représentant un agent correspond à une caractéristique.
Vu que le problème que nous résolvons est la variante la plus simple mais en même temps la base, ces caractéristiques sont donc obligatoires.

Caractéristiques de l'Interaction : Les caractéristiques de l'interaction dépendent des types d'applications et par conséquent de la *plateforme*. Dans les SMA, il existe deux types d'interactions : L'interaction directe et l'interaction indirecte. Par exemple, JADE utilise l'interaction directe et fournit une librairie de protocoles d'interaction - chacun étant implémenté par deux rôles d'interactions. Dans le diagramme de classes, les classes *CNP-Initiator* et *CNP-Participant* représentent les deux rôles du protocole *Contract Net* (CNP). Par ailleurs, Netlogo utilise souvent l'interaction indirecte telle que les phéromones.

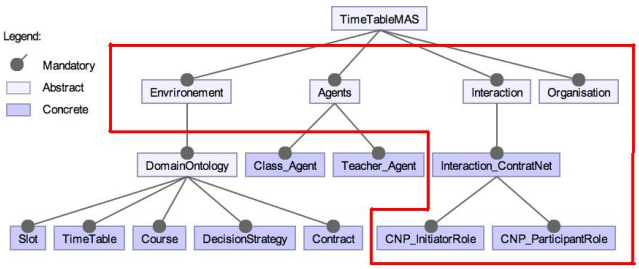

FIG. 4 – Modèle initial des caractéristiques pour le SMA de l'emploi du temps.

Puisque dans notre système l'utilisation de CNP est nécessaire, nous proposons d'associer la caractéristique *Interaction-ContractNet* au comportement *FSM_Behaviour*, avec ses sous-caractéristiques *CNP_InitiatorRole* et *CNP_ParticipantRole* pour chaque classe qui décrit les rôles d'interaction.

Caractéristiques de l'organisation : Compte tenu de la simplicité de la première version de notre exemple, il n'y a qu'un groupe d'agents (comprenant un agent Enseignant, *n* agents Classes) et deux rôles. Chaque agent joue un rôle. Par conséquent, nous n'avons pas nécessairement besoin d'un modèle explicite de l'organisation pour cette version.
Une capture d'écran du FM est donnée dans la figure 4, et elle représente une configuration du SMA-LdP pour l'exemple décrit dans la section 3. Les caractéristiques entourées en rouge sont indépendantes du domaine et représentent le FM. Le reste des caractéristiques est spécifique au domaine. Ce FM sera raffiné afin de montrer explicitement la variabilité associée à certaines variantes de notre exemple.

4.2 L'évolution des SMA par le raffinement des caractéristiques

Etant donnée la solution développée dans la section précédente, nous allons analyser et développer quelques extensions en améliorant le FM. Nous proposons d'augmenter la complexité du problème :

- en changeant la stratégie de décision pour les deux rôles. Dans notre premier FM, nous considérons que les différents agents sont homogènes. Ils utilisent la même stratégie pour sélectionner les propositions reçues. Les classes utilisent la même stratégie pour accepter ou rejeter les propositions, et tous les enseignants utilisent la même stratégie pour sélectionner un sous-ensemble de propositions à accepter ;
- en changeant le nombre d'agents Enseignants dans notre système (Version 1 : 1 Enseignant et n Classes ; Version 2 : m Enseignants et n Classes). Ainsi, les agents Classes peuvent recevoir plusieurs propositions pour le même cours. Ensuite, les agents Classes peuvent définir une stratégie pour sélectionner l'un d'entre eux à accepter. Cela signifie que nous devrions étendre le protocole d'interaction pour décrire cette nouvelle façon d'interagir ;

- en ajoutant des contraintes sur les ressources telles que la disponibilité des salles. Nous pouvons ainsi ajouter un agent pour gérer toutes les salles ou associer à chaque salle un gestionnaire ;
- en utilisant un autre protocole d'interaction.

Pour changer de protocole, nous avons juste besoin d'étendre le FM existant en incluant une caractéristique abstraite Interaction-Protocol qui est composée de plusieurs protocoles d'interaction, tels que les protocoles de FIPA *Contract-Net*, *Iterated-Contract-Net*, *English-Auction*, *Request-Interaction*, entre autres. Nous avons ensuite analysé les protocoles d'interaction et les comportements requis. Le but de cette analyse est de déterminer les comportements qui peuvent être génériques. Par exemple, l'initiateur CNP nécessite un contrat, un temps d'arrêt (*timeout*) et une stratégie d'évaluation. En outre, le participant exige seulement une stratégie pour construire des propositions. Jarraya et Guessoum [13] concluent que, pour plusieurs protocoles d'interaction, les comportements locaux (définis comme automates à états finis) sont génériques. Mais ils nécessitent certains paramètres d'entrée comme un contrat, un temps d'arrêt, et une ou plusieurs stratégies.

En effet, ils peuvent être adoptés et activés dynamiquement par des agents.

Nous proposons donc d'ajouter à notre FM un ensemble de caractéristiques représentant les rôles des protocoles d'interaction, et un ensemble de stratégies, qui enrichissent l'ontologie. En outre, et pour gérer les salles de classe et leur disponibilité, nous proposons d'ajouter le concept *Room* à l'ontologie et de créer une nouvelle classe représentant les gestionnaires (*Room_Agent*).

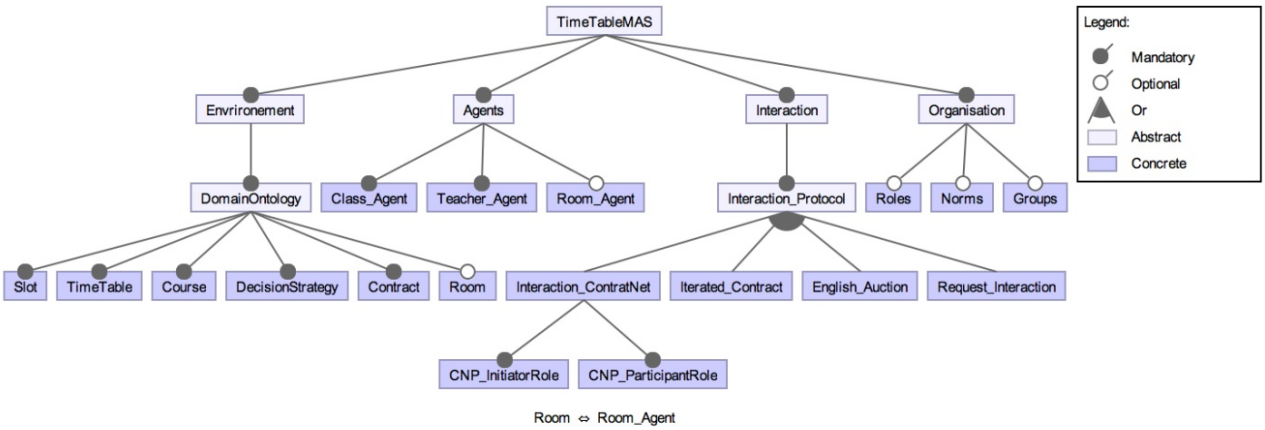

FIG. 5 – Le modèle des caractéristiques raffiné.

La Figure 5 illustre le nouveau FM qui raffine le modèle initial de la Figure 4. Il est à noter que les modèles de caractéristiques permettent également d'ajouter les dépendances entre les caractéristiques.

Nous ajoutons donc une contrainte spécifiant que la présence de la caractéristique liée à la ressource *Room* nécessite la présence de la caractéristique *Room-Agent* et vice versa.

Il est important de noter également que dans cet article nous ne considérons pas la caractéristique de l'organisation qui concerne cet exemple d'emploi du temps. En plus, les comportements des agents sont limités à des rôles interactifs et aucune norme n'est considérée. Toutefois, si l'on considère un problème plus complet pour gérer l'emploi du temps des collègues de l'Université, plusieurs aspects doivent être considérés. Par exemple, plusieurs normes doivent être prises en ligne de compte comme le nombre d'heures minimal ou maximal. Par ailleurs, un modèle d'organisation tel que AGR [11] devrait être considéré également. Nous avons donc ajouté trois caractéristiques optionnelles : les Rôles, les Normes et les Groupes.

4.3 Implémentation d'artefacts et dérivation de variantes

Comme souligné dans la section 2, la LdP est

définie par un FM mais aussi par des éléments logiciels (appelés *assets*). Ces derniers représentent les artefacts logiciels qui implémentent les différentes caractéristiques. De nombreuses approches ont été proposées pour mettre en œuvre les artefacts logiciels dans les LdPs [2]. Dans notre approche où l'objectif est de générer des applications JADE, nous nous sommes intéressés aux artefacts logiciels représentant des fragments de code JADE. Pour implémenter ces artefacts, nous proposons de réutiliser le composeur FeatureHouse [1] qui permet l'implémentation de la LdP en Java. FeatureHouse associe à chaque caractéristique les éléments de code l'implémentant. Ceci peut contenir l'ajout de nouvelles classes et/ou le raffinement des classes existantes. FeatureHouse utilise par la suite un mécanisme de composition pour générer le code de chaque variante en se basant sur une composition des caractéristiques [1].

Dans ce qui suit, nous illustrons l'implémentation des caractéristiques pour l'emploi du temps. Ensuite, nous présentons la façon dont chaque variante SMA peut être dérivée à partir de cette implémentation.

4.3.1 Implémentation des artefacts

Comme indiqué ci-dessus, les caractéristiques LdP-SMA peuvent concerner quatre dimensions : l'environnement, les agents, l'interaction et l'organisation. Les caractéristiques qui concernent l'environnement pour le SMA de l'emploi du temps telles que *TimeTable*, Course*, Slot, et Contract*, sont directement mappées aux classes Java.

L'implémentation des caractéristiques liées aux agents est basée sur l'extension des classes agents de JADE. Par exemple, le code associé à JADE pour la classe *Teacher-Agent* consiste à définir la classe de façon à ce qu'elle hérite de la classe *Agent* de JADE. Nous avons également besoin de raffiner la classe *Launcher* en ajoutant l'initialisation de l'agent *Teacher-Agent*. En effet, pour chaque implémentation JADE mise en œuvre, une classe de lanceur devrait être définie pour initialiser les différents agents.

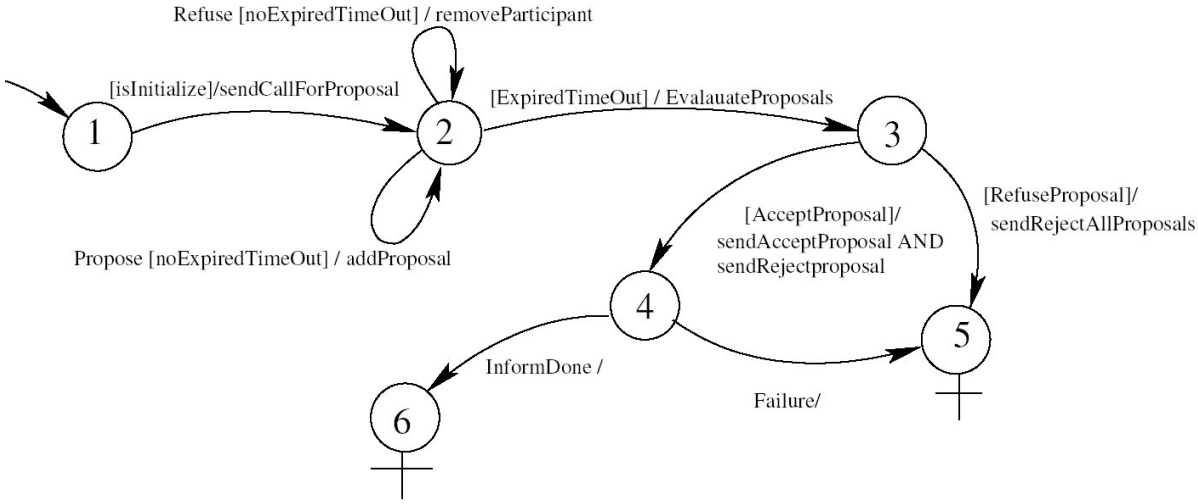

FIG.6 – Machine d'états finis représentant le rôle initiateur du CNP.

La mise en œuvre des caractéristiques *CPN-Initiator* et *CPN-Participant* qui sont les deux rôles liés au CPN, est basée sur la réutilisation de la classe *jade.core.Behaviour* de JADE. Pour rendre le comportement associé générique, nous utilisons le *FSMBehavior*. Chaque rôle est donc mis en œuvre comme une sous-classe de cette classe.

La Figure 6 donne un exemple de ce FSM à travers sa machine d'état associée. Elle s'appuie sur un contrat, une stratégie d'évaluation, une liste de participants et ne nécessite pas une autre entrée pour être exécutée.

4.3.2 Dérivation de variantes

A partir du modèle des caractéristiques des SMA et de tous les objets logiciels associés, différentes variantes peuvent être dérivées automatiquement sur la base des mécanismes de composition, comme ceux définis dans FeatureHouse [1].
La dérivation des variantes est implémentée en deux étapes :

- créer une configuration valide du FM qui contient la sélection des fonctionnalités pour une variante spécifique,
- composer les artefacts logiciels qui sont associés à la fonction sélectionnée, et ce, à l'aide du compositeur.

Pour le SMA de l'emploi du temps, plusieurs configurations valides peuvent être créées à partir des caractéristiques du modèle de la figure 5.

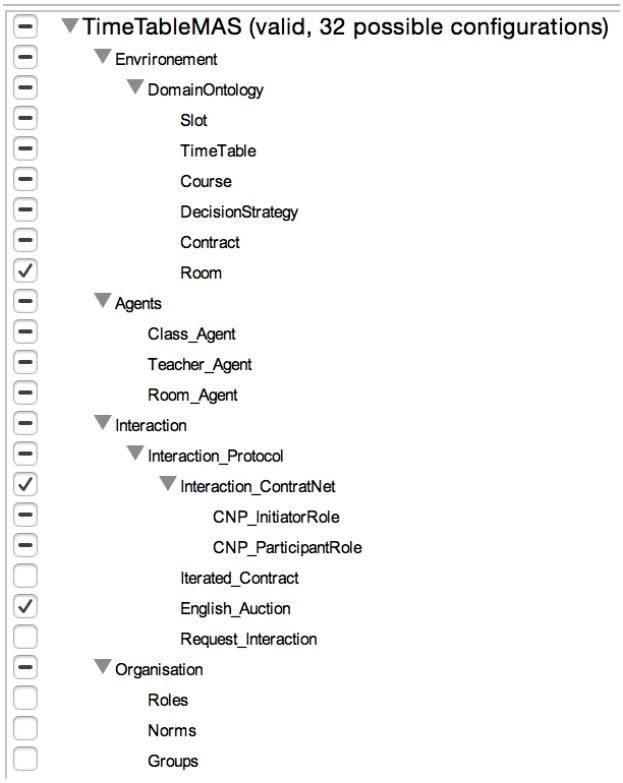

FIG.7 –Exemple de configuration valide pour une variante de l'emploi du temps.

La figure 7 montre un exemple d'une telle configuration valide qui concerne une variante du SMA de l'emploi du temps supportant deux types de protocoles d'interaction : *ContratNet* et *English_Auction*. Cette variante gère également les contraintes de salles parce que les deux caractéristiques *Room* et *Room-Agent* sont sélectionnées. Cependant, toutes les caractéristiques qui sont liées à la dimension Organisation ne sont pas présentes.
De la configuration valide créée, FeatureHouse compose le code Java pour toutes les caractéristiques choisies afin de générer le code source de la variante cible.
La génération de code est basée sur le raffinement du code [1].

5. Conclusion et perspectives

Nous avons introduit dans ce papier une approche LdP-SMA, nommée Id-LdPMAS qui permet le développement incrémental basé sur les caractéristiques et leur implémentation, et qui comble le fossé existant entre la modélisation et l'implémentation des SMA. Pour outiller cette approche, nous avons utilisé FeatureHouse, un framework de LdP qui permet de générer des variantes du système à partir des caractéristiques. Pour l'implémentation des SMA, nous avons utilisé JADE. Chaque caractéristique est associée au code JADE et/ou Java.
Id-LdPMAS utilise un ensemble de caractéristiques qui peuvent être déduites de quelques modèles SMA existants. L'identification de ces caractéristiques et leur catégorisation sont réalisées sur la base du paradigme Voyelles. Par conséquent, l'approche pourrait être adoptée par les développeurs utilisant n'importe quelle méthode pour créer des modèles de SMA, y compris les méthodes situationnelles [7].
Id-LdPMAS a été validé sur l'exemple de l'emploi du temps avec la plate-forme JADE. En perspectives nous projetons : (i) d'analyser l'impact sur l'identification des caractéristiques lors de changement de plateforme ; (ii) de considérer d'autres exemples de SMA qui s'articuleraient autour de modèles organisationnels tel que celui des fourmis et du champs potentiels afin d'enrichir le modèle avec différents mécanismes d'interaction et d'auto-organisation également ; (iii) de

proposer des lignes directrices et des tutoriels accompagnés d'exemples pour les développeurs des applications SMA, afin de les encourager à adopter les LdP-SMA en les aidant à construire leur propre LdP-SMA étape par étape ; (iv) de proposer l'adoption de l'agilité tout en la combinant avec notre approche, ce qui serait intéressant pour concevoir une véritable méthode pour les SMA.

Remerciements

Anarosa A. F. Brandão est financée par une bourse #014/03297-7, São Paulo Research Foundation (FAPESP).

References

[1] S. Apel, C. Kästner, C. Lengauer, Language-Independent and Automated Software Composition: The FeatureHouse Experience. *IEEE Transaction on Software Eng*ineering 39(1): 63-79, 2013

[2] S. Apel, D. Batory, C. Kästner, G. Saake: *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, 2013.

[3] D. Benavides, S. Segura, and A. R. Cortés, Automated analysis of feature models 20 years later: A literature review, *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[4] F. Bergenti, M-P. Gleizes and F. Zambonelli, *Methodologies and Software Engineering for Agents Systems*, Kluwer Publishing, 2004.

[5] C. Bernon, M-P Gleizes, P. Glize, G. Picard, Le problème de l'emploi du temps - Cahier des charges, In C. Bernon, V. Camps, M-P. Gleizes, P. Glize, S. Peyruqueou, G. Picard. Ingénierie des AMAS pour l'emploi du temps. ETTO - Emergent TimeTable Organization. *Rapport de recherche, ASA-PRC IA, IRIT*, 2002.

[6] S. Casare, A. A. F. Brandão, Z. Guessoum, J. Sichman, Medee Method Framework: a situational approach for organization-centered MAS. Autonomous Agents and Multi-Agent Systems, 28(3): 430-473, 2014.

[7] E. Cirilo, I. Nunes, U. Kulesza, C. Lucena, Developping Multi-Agent System Product Lines: From Requirements to Code. *IJAOSE*, pp. 197-216, 2011.

[8] Clements and L. Northrop, *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., 2001.

[9] J. Dehlinger, R. Lutz, Gaia-PL: A Product Line Engineer-ing Approach for Efficiently Designing Multi-Agent Systems. *ACM Transaction on Software Engineering Methodol*ogies 20(4): 17 (2011)

[10] Y. Demazeau, From interactions to collective behavior in agent-based systems. *Proceedings of the 1st. European Conference on Cognitive Science*. Saint-Malo, p. 117-132, 1995.

[11] J. Ferber, O. Gutknecht, F. Michel, From Agents to Organizations: An Organizational View of Multi-agent Systems. In P. Giorgini, J.P. Müller, J. Odell (Eds.): *Agent-Oriented Software Engineering*, LNCS 2935, pp. 214–230, Springer, 2004.

[12] Z. Guessoum, M. Cossentino and J. Pavon Mestras. A Roadmap of Agent-Oriented Software Engineering: The European Agentlink Perspective. In "*Methodologies and Software Engineering for Agents Systems*", Bergenti ey al (eds.), Kluwer pp. 430-450, 2004.

[13] T. Jarraya, Z. Guessoum, Towards a Model Driven Process for Multi-Agent System. *Proc.of CEEMAS 2007*: 256-265

[14] K. Kang, S. Cohen, J. Hess, W. Nowak, S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study (Report). *CMU/SEI-90-TR-21*.

[15] I. Nunes, C. Lucena, D. Cowan, U. Kulesza, P. Alencar, C. Nunes, Developing Multi-agent System Product Lines: From Requirements to Code, Agent-Oriented Software Engineering, 4(4), 353-389, 2011.

[16] J. Peña, M. Hinchey, M. Resinas, R. Sterritt, J. Rash, Designing and managing evolving systems using a MAS product line approach. Sci. Comput. Program. 66(1), 71-86, 2007

[17] J. Peña, M. Hinchey, A. Ruiz-Cortés, Multi-agent system product lines: challenges and benefits. Communications of the ACM, Vol. 49 No. 12, Pages 82-84. 2006.

[18] M. Cossentino. N. Gaud, V. Hilaire, S. Galland, A. Koukam. ASPECS: an agent-oriented software process for engineering complex systems. In Autonomous Agents and Multi-Agent Systems, 20(2), 260-304, 2010.