# Towards a MAS Product Line Engineering Approach

Dounia Boufedji, Zahia Guessoum, Anarosa Brandão, Tewfik Ziadi, Aïcha Mokhtari

# Towards a MAS Product Line Engineering Approach

Dounia Boufedji[13], Zahia Guessoum[12], Anarosa Brandão[4], Tewfik Ziadi[1], and Aicha Mokhtari[3]

[1] LIP6, UPMC Paris 06 University, France
[2] CReSTIC, Reims Champagne Ardenne University, France
[3] RIIMA, USTHB Sciences and Technology University, Algeria
[4] Computing Engineering&Digital Systems Department, São Paulo University, Brazil

**Abstract.** It is our claim that the adoption of software engineering reuse techniques can leverage MAS development, mostly when we consider similar applications belonging to the same domain. MAS-Product Line (MAS-PL) raises as an interesting approach that uses Software Product Line Engineering (SPLE) techniques and AOSE to manage the commonalities (similarities) and variabilities (differences) of such MAS applications. Although MAS present specific characteristics that could be considered when describing the system variability, existing work on MAS-PL is devoted to deal with MAS variability considering only domain-specific issues. Moreover, the adoption of variability models such as feature models should be considered for describing both Generic and Specific MAS variability. We propose a MAS-PL approach to address the aforementioned issues by representing Generic MAS variability according to MAS concepts such as agents, environment, interaction and organization, and Specific MAS variability according to a specific application domain.
We evaluate the approach by deriving a family of agents that perform jobs in the Multi-Agent Contest environment.

**Keywords:** Software Product Line Engineering, Software Engineering Reuse, Feature Model, Variability.

## 1 Introduction

Multi-Agent Systems (MAS) provide an interesting approach for developing software systems in several domains such as resource and information management, process control and simulation of complex systems. Nevertheless, since engineering MAS is a complex task that is not entirely controlled by a process which is accepted by the software industry, MAS are still not part of the mainstream of enterprise application development [17]. In this paper, we propose to introduce a new MAS-PL approach to improve the reuse during MAS development.

AOSE provides several templates and reuse patterns to facilitate MAS development [10, 18] and speed up the MAS adoption in software industry. However,

most existing AOSE approaches are not suited to the development of similar applications (a.k.a MAS families). These kinds of applications present similarities (i.e. commonalities) and differences (i.e. variabilities).

Indeed, managing such applications remains a difficult task because even if the core architecture is reusable, the variability management is mainly achieved by making code changes.

As the facts mentioned above lead to a considerable waste of time, cost and effort, and make the MAS implementation a difficult task, it is important to provide a solution based on the idea of capitalizing the MAS implementation expertise. Thus, managing MAS variability in MAS families is a key solution to such a capitalization, and has become one of the main challenges in AOSE. SPLE comes out as an interesting solution to manage MAS variability at different levels such as in design models, implementation of agents and so on. It provides a solution to speed up industrial adoption of MAS, and leverages such an adoption [23].

Several MAS-PL approaches have been proposed [3, 11, 21, 22] to apply SPL concepts to MAS development. Those MAS-PL approaches introduce the notion of variability for a reuse issue within a MAS family. They are often built as an extension of existing methods, what make them easily adopted. However, most existing approaches introduce different notations and stereotypes to specify variability, what concerns only specific domains.

It is our claim that a MAS-PL approach should address both the variability concerning a specific application domain, and the variability referring to the MAS domain, which concerns MAS concepts that emerge from MAS meta-models and tools.

In addition, we believe that specifying variability by using only variability models (e.g. feature model) is more interesting than using a variety of models (e.g. roles' variation model) or introducing new notations (e.g. extensions of MAS design models) for the same purpose. Example of that is proposed by Peña et al. [24], where three kinds of models are used to specify variability: feature model, roles variation model, and plans variation model. In our view, roles and plans variation could be specified using a single variability model, such as a feature model.

We propose a new MAS-PL approach that follows the general SPLE Framework [1]. This approach relies on two types of features (resp. two types of reusable artifacts): the *Generic MAS* features (resp. artifacts) and the *Specific MAS* features (resp. artifacts). This will concretely serve to promote different types of reuse in MAS.

Those features result from a refinement process that is based on a variability analysis of distinct domains. *Generic MAS* variability analysis scopes the domain of existing MAS methods, meta-models, architectures and tools, while *Specific MAS* variability analysis scopes a specific application domain.

This paper is organized as follows: Section 2 presents a background, dealing with MAS-PL related work and motivations. Section 3 gives an overview of our MAS-PL approach; while Section 4 and Section 5 give more details about the

approach and illustrate it with simple examples of the multi-agent contest case study. Section 6 describes and discusses some results about the evaluation of our approach through the multi-agent contest product line. Finally, Section 7 summarizes the contributions and proposes some perspectives for future work.

## 2 Background and Related Work

### 2.1 Software Product Line Engineering framework

SPLE represents one of the most interesting paradigms in software reuse and development. It reconciles both production and standardization with customization in software engineering, and it considerably reduces development cost, time and effort. The general SPLE framework proposed by Apel et al. [1] is represented in Fig. 1 with its two levels: Domain and Application Engineering. The framework includes four activities: domain analysis, domain implementation, requirement analysis, and product derivation.

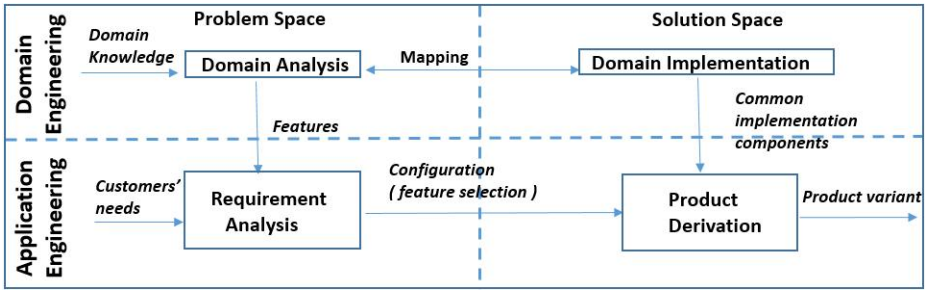


**Fig. 1.** SPLE Framework and its main activities. [1]

**Domain Analysis** allows to scope the domain (which products should be covered by the product line), and to specify the relevant features that should be implemented as reusable artifacts. The results of domain analysis are usually documented in a Feature Model (FM) [1]. The FM describes features and their relationships (parent-child) in a hierarchical tree. Features express the commonality and variability among the products within a product line (see examples in Fig. 3).

**Domain Implementation** allows to develop reusable artifacts that correspond to identified features. There are many kinds of relevant artifacts in SPLE (including implementation, test, and documentation artifacts) [1].

**Requirement Analysis** considers a user's requirements to produce a customized configuration, by selecting the desired features.

**Product Derivation** aims at deriving the product according to the configuration provided by the requirement analysis.

### 2.2 Related Work

MAS-PL approaches emerged from the idea of applying SPLE approaches into AOSE ones. The first efforts on MAS-PL arose on the mid of the 2000's first decade. Peña et al. [23] argue that both AOSE and SPLE approaches are based on similar concepts in the first activities of domain engineering. For instance, both of them use models in the domain analysis activity: SPLE uses feature models and AOSE uses MAS meta-models. Unlike AOSE, SPLE covers common and variable features analysis of the software family. Most existing AOSE approaches do not consider implementation activity, while SPLE also relies on implementing reusable assets.

MAS-PL approaches differ according to the SPLE phases they cover: Domain and Application Engineering. Most approaches propose to extend AOSE methods by integrating SPLE concepts and techniques.

In this context, Dehlinger et al. [11] propose Gaia-PL to extend GAIA. They provide requirement specification pattern to capture changing design configuration (variation points) in agents and potential reuse of requirement specification; with no detail about the domain implementation activity.

Nunes et al. [21, 22] propose to extend PASSI [9], to cover the whole development process from requirements to code. They endow PASSI's UML models with stereotypes to model and document agent variability, and they propose implementation guidelines to help MAS developers. Moreover, they propose the modularization of the fine-grained variability (agent architecture) allowing a better specificity of the features [20]. For instance, they introduce decomposition of the goals that gives more specific plans. Although the decomposition of goals ensures alternative or optional features, reuse is only possible for a specific domain application.

Peña et al. [24] propose to enrich MaCMAS (Methodology for analyzing Complex Multi-Agent Systems) with software product lines to model and evolve MAS. They use UML to model a MAS-PL and focus on building the core architecture (common features). MaCMAS captures views of the system at different abstraction levels. The core architecture of the system is represented by a traceability model, and a set of role models. The model is evolved with variations and constraints. They specify the commonality and variability in a feature model.

MAS-PL approaches that cover the Application Engineering phase propose mainly MAS derivation approaches that extend derivation tools like in [8] where they propose to use multi-level models to support the configuration knowledge specification and automatic product derivation of MAS-PL.

Among MAS-PL approaches that cover both SPLE phases, SelfStarMAS proposes a process for the development of self-adaptive agents in Internet of Things (IoT), and extends it by a Dynamic SPL based approach, that presents the advantage of behaviour agent adaptation at runtime [3]. The approach is interesting, but remains specific to IoT domain.

Even though many approaches have been proposed, some limitations can be clearly identified:

- The use of multiple notations and stereotypes in UML diagrams to model variability is more difficult to adopt than using feature modeling notations only, what is more recommended by the SPLE [1];
- Using colors while introducing crosscutting features and spreading variability specification along different kinds of models may prevent their adoption, since any change on features must be propagated to all models;
- Some approaches focus on building the core architecture of MAS families and neglect variabilities. Some others focus on the domain analysis activity and do not give details on the domain implementation;
- Reusing feature models and artifacts, when developing a new family of applications, is unfeasible since the specified variability is domain-specific in all approaches.

As a solution to those limitations, we propose a new MAS-PL approach with one category of variability models: feature model; and two kinds of features: generic MAS features and application specific features. In the next section, we introduce our MAS-PL approach.

## 3 Overview of our MAS-PL approach

Our approach proposes to develop MAS families by following the general SPLE framework [1] and reusing MAS concepts (see Fig. 2).

Our approach splits the Domain Analysis (resp. Domain Implementation ) activity of the domain engineering phase into two activities: 1) Generic MAS domain analysis (resp. Generic MAS domain implementation) and 2) Specific MAS domain analysis (resp. Specific MAS domain implementation). This distinction between *Generic MAS* domain and *Specific MAS* domain aims at capitalizing the expertise of using MAS methods, models and implementations, by reusing both *Generic MAS* features and *Domain specific* ones.

First, during the *Generic MAS* domain analysis activity, we analyze the MAS knowledge in terms of generic MAS concepts which are involved in MAS approaches like GAIA [7], and PASSI [9]. Then, we represent *Generic MAS* concepts in terms of commonalities (similarities) and variabilities (differences) among MAS approaches, organized so that they refer to *Agent*, *Environment*, *Interaction* and *Organization* features. Our approach relies on AOSE methods, to enable reusing most common concepts and assets, and to provide flexibility to MAS designers and developers by using the features they need. For instance, the *Role* concept that is provided by GAIA [7], PASSI [9] and AGR [14] can be placed as a child-feature component of the *Organization* feature. To illustrate the activity, we analyze the belief-desire-intention (BDI) model [16], some organizational models like Moise+ [19], and so on. This activity produces a *Generic MAS FM*.

Second, *Specific MAS* domain analysis activity documents similarities and variabilities among the members of a specific MAS family. We illustrate this activity with the multi-agent contest example [1]. This activity produces a *Specific*

[1] https://multiagentcontest.org/

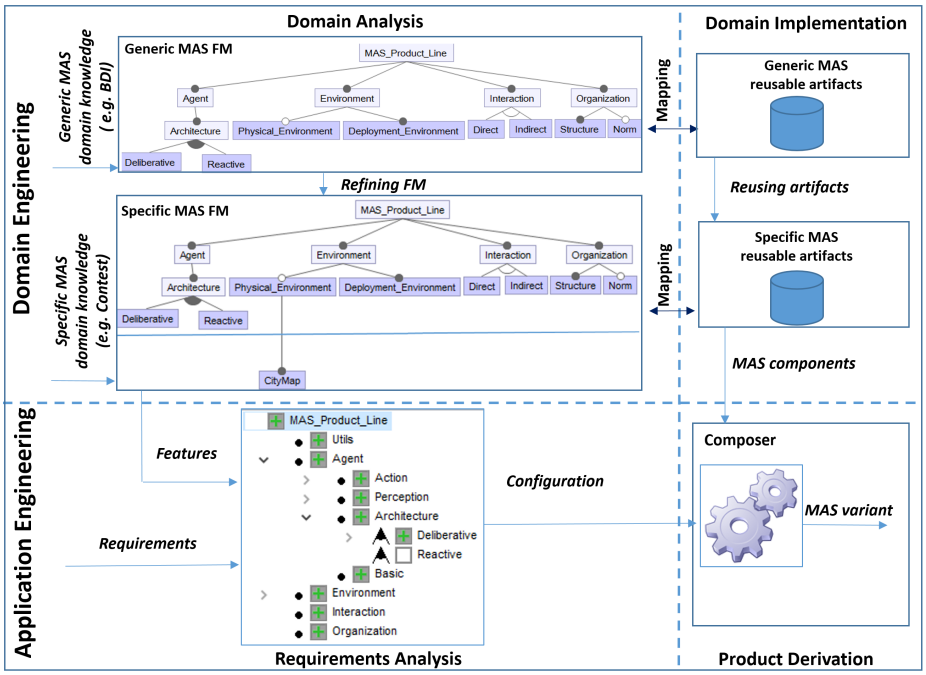


**Fig. 2.** Overview of our MAS-PL approach

*MAS FM*. Unlike existing MAS-PL approaches that build MAS specific FM from scratch, our approach proposes to build it by reusing the *Generic MAS FM*.

Third, *Generic MAS* implementation activity involves implementing *Generic* reusable MAS artifacts, that do not rely on any specific MAS family. They implement *Generic MAS* features and are composed of Agent, Environment, Interaction and Organization artifacts. Those artifacts often come from existing tools and frameworks.

Finally, *Specific MAS* implementation activity produces *Specific MAS* reusable artifacts. *Specific MAS* implementation relies on *Generic MAS* implementation. Artifacts are adapted to the specific MAS family.

Application engineering activities concern both MAS requirement analysis and derivation. According to the MAS requirements, MAS variants are specified by selecting a valid configuration from the *Specific MAS FM*. The specified product that represents the MAS application variant is then derived. As we are interested in *Composition-Based* implementation approaches [1] instead of *Annotation-Based* ones, we need a *Composer* to derive the product. This derivation activity is done by FeatureHouse composer [2] that generates MAS variants automatically by composing reusable artifacts. We illustrate our approach by Contest Agent Variants based on configurations produced during the Application Engineering phase.

# 4 MAS Domain Engineering

This section describes the proposed domain engineering phase based on activities and their outputs.

## 4.1 Generic MAS domain analysis

In this first activity, we analyze MAS models' commonalities and variabilities to build the *Generic MAS FM* which is a compact representation of MAS concepts. We therefore define the features that are often shared by MAS applications.

Our idea is to produce a common FM (see Fig.3) based on that generic representation and to use the Vowels paradigm [12] to organize these features. The Vowels paradigm considers that MAS are composed of (1) Agents (Vowel A), which refers to the description of internal architectures of the system processing entities; (2) Environment (Vowel E), which refers to domain-dependent elements for structuring external interaction among the system entities; (3) Interaction (Vowel I), which refers to elements for structuring internal interaction among the system entities; and (4) Organization (Vowel O), which refers to elements for structuring entities within the MAS. We do not include the User (Vowel U) dimension that is considered in the Vowels extension to explicitly take into account the user.

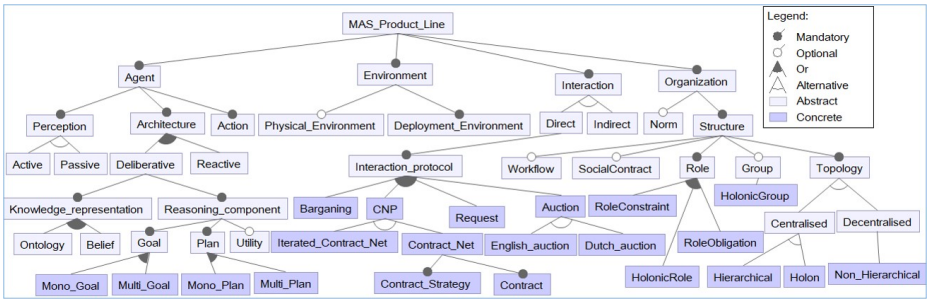


**Fig. 3.** Generic MAS Feature Model

**Agent features:** Agent architectures provide solutions to structure agents and define their functionalities in order to enable them to act and to interact in a dynamic environment. Most existing architectures follow the perception-action loop. We thus propose three categories of features: *Perception*, *Architecture* (Internal Architecture) and *Action*. For the Internal Architecture, we consider both categories: *Reactive* and *Deliberative* [13]. While deliberative agents follow Perception-Deliberation-Action cycle, reactive agents follow Perception-Stimulus-"Re"action one. *Hybrid* architectures can also be defined by combining the previous ones (selecting both *Deliberative* and *Reactive features*).

Deliberative agents need a knowledge model to provide them a representation about their environment, and their own knowledge. The *Knowledge_representation* feature represents knowledge representation such as *Belief* and *Ontology*.

The BDI model [16] includes three mental attitudes: Beliefs, Desires and Intentions. Whenever the agent has a BDI architecture, all of *Belief*, *Goal*, and *Plan* features are mandatory.

The *Utility* feature of our *Generic MAS FM* represents an option used to endow Goal_Based agents with an utility measure for evaluating the level of success when reaching the goal, to obtain Utility_Based agents.

**Environment features:** Agents are situated in an environment that used to be domain dependent and generally spatial. That environment represents many aspects that conceptually do not belong to agents themselves such as in a software infrastructure on which the MAS is deployed, or in a representation of physical environment. City maps used for situated agents can be mentioned as an example. We represent the agent Environment product line by two environment features: the *Deployment_Environment* which is mandatory and the optional *Physical_Environment* since the physical world is not always represented in MAS.

**Interaction features:** Interaction provides a way to ensure coordination of agents' activities. Agents' interaction can be *Direct* or *Indirect*. In direct interaction, agents exchange messages to coordinate their behaviour and achieve the global goal. In indirect interaction, agents use the environment to share information and coordinate their actions. For example, ants use the pheromone to coordinate. In this paper, we focus only on direct interaction that is regulated by interaction protocols. The *Interaction_protocol* feature is therefore mandatory. Interaction protocols were introduced into MAS to facilitate the specification and the implementation of interaction between agents. According to FIPA [2] definition, an interaction protocol is a common pattern of communication (a predefined sequence of messages). Thus the specification and the implementation of the Protocol could be independent of the scope and of the agent internal architecture. Several interaction protocols have been proposed: request protocol, bargaining, auction and Contract Net Protocol (CNP), among others. Our *Generic MAS* FM includes some of them with a possible feature selection.

**Organization features:** In MOISE+ model [19], the organization is seen under three points of view: Structural, Functional, and Normative. We propose to model the MAS Organization product line by the root *Organization* feature that includes two-child features: the *norm* optional feature; and the mandatory *structure* feature to represent all possible organization structures.

Ferber et al. proposed the first organizational model Agent-Group-Role (AGR) [14], to highlight the importance of organizational concepts like 'groups'. The *Role* concept is used in most existing MAS organizational meta-models while the *Group* concept is used only in some of them such as AGR and AGRE [15]. So, the *Group* feature is optional while the *Role* feature is mandatory.

[2] http://fipa.org/

The last mandatory feature that concerns the *structure* is the *Topology*, that has two alternative child features: *Centralized* and *Decentralized* organizations described with other child features.

### 4.2 Specific MAS domain analysis

During this activity, we analyze MAS commonalities and variabilities of a specific MAS family (e.g. Multi-Agent Contest) to produce a *Specific MAS FM* to refine the *Generic MAS FM*. The refinement process is achieved by adding Specific features to solve the problem. These specific features are placed hierarchically under the generic features they are specializing.

We propose as an example, the Multi-agent Contest specific domain in order to illustrate the rest of the activities of our approach. We will refer to *Specific MAS* features as *Contest MAS* features. Contest agents' teams move around the streets of a realistic city, having the goal of earning money by completing jobs. Teams should then decide how to navigate on the city map, and where to get the resources to assembly, buy and deliver items considering targets like shops, warehouses, charging stations, and storage facilities. Tournament points are distributed according to the amount of the money a team owns at the end of the simulation.

Fig.4 depicts three examples of possible *Contest MAS FM*. The two categories of features are separated by a red line. The examples show what MAS product line designers should do to refine the *Generic MAS FM*. However, before doing it, *Contest MAS* features should be detected by the domain variability analysis.

For instance, since the provided Contest environment is mandatory, the Contest mandatory feature *cityMap* should be added to Contest features. After, the corresponding *Generic MAS feature* is refined. The result is presented in Fig. 4 on (c) where *cityMap* refines the *Physical_Environment* feature. Another example represented on (a) concerns Agent variability. The specific *BuyItem_Goal* feature refines the *Mono_Goal* for specific items acquisition goal. This first version of the *Specific MAS FM* represents the simplest one, and considers only agents that achieve one goal. But, as our approach is incremental, this simplest version of the *Specific MAS FM* can be also refined in turn. The left side of Fig. 5 depicts another version of the *Specific MAS FM* which refines the simplest one. This refinement allows to support other versions of agents. For example, both of *DeliverItem_Goal* and *Charging_Goal* Contest specific optional Features refine the generic *Multi_Goal* feature by specifying agents that have to achieve items delivering and charging goals. When considering other generic variabilities such as Organization ones, the process remains the same. The last example (see Fig.4 on (b)) proposes possible roles of the organization of a team in Contest. This aspect considers the structural point of view of the organization, and is relative to the *Structure* feature. Thus, the *Role* generic MAS feature is refined by Contest specific roles such as *Buyer* and *Carrier* that will take part in all Contest Team Variants. While *TruckGroup* feature is an option to consider groups' organization.

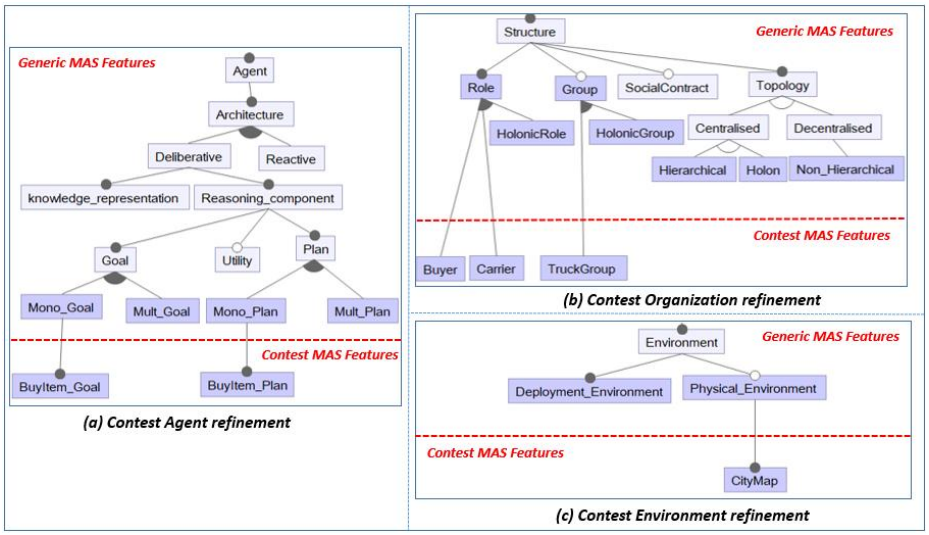


**Fig. 4.** Examples of the Contest Specific MAS FM that refines the Generic MAS FM: (a) Contest Agent refinement, (b) Contest Organization refinement and (c) Contest Environment refinement

### 4.3 Generic MAS Implementation

The following activity concerns the *Generic MAS reusable artifacts* implementation. The MAS implementation is often based on existing frameworks. Artifacts correspond to components provided by those frameworks. In this paper, we consider APLTK (A Toolkit for Agent-Oriented Programming) [4] to illustrate the feasibility of the approach in the implementation side. The set of reusable artifacts can be then enriched by considering other frameworks such as JaCaMo [6].

**Agent artifacts:** Agent reusable artifacts implement *Agent* features. Although we could not raise variability concerns related to *Belief*, both *Goal* and *Plan* can present variability depending on the choosing BDI algorithms.

The three algorithms we use to illustrate our approach are those proposed by Wooldridge [16]. The variability among these algorithms lies on the cardinality of the sets of B, D and I. The first algorithm represents the simplest version. It corresponds to mono-Goal agents. While the second algorithm concerns agents that have to achieve multiple goals. The last algorithm proposes to enrich the library of plans during the execution.

*Goal* and *Plan* variabilities can then be expressed by the following features: *Mono_Goal*, *Multi_Goal*, *Mono_Plan* and *Multi_Plan*.

Most BDI model implementations use *brf* (belief revision function), *ogf* (option generation function), *filter* and *asf* (action selection function). However, the algorithm variability has an impact on these functions. For instance, if we

consider BDI *Mono_Goal* implementation, agents use neither the *ogf* function nor the *filter* one. *Mono_Plan* agents do not need a function to select a plan.

We implement the *Mono_Goal* feature with reusable artifacts, composed of the functions proposed in the simplest algorithm of Wooldridge [16] : (i) *getting-Percepts()*: to execute the get-next percept, (ii)*createBeliefsFromPercept()*: to create and update the agent beliefs, it represents the *brf()* function; (iii)*checkAll-BeliefsForInsertGoal()*: for the agent deliberation to correspond to the *deliberate()* function, and (iv)*performActionGoal()*: to select a plan and execute it.

**Environment artifacts:** The general view of the environment considers that agents are part of the environment, which can provide for example means or resources for agent communication. However, environment standardization should be done by separating both concepts. Behrens et al. [5] proposed a generic approach for connecting agents to environment, and considered reusable environment artifacts that are as much independent of a specific environment structure as possible. For example, we reuse EIS (Environment Interface Standard) API [5], that represents possible reusable environment artifacts, which are mapped to the *Physical_Environment* feature. EIS reduces the implementation effort for connecting to the environments (e.g. Unreal Tournament UT3 and UT2004 gaming environments, and the Multi-Agent Contest).

**Interaction artifacts:** The reusable Interaction artifacts concern mainly the Interaction protocols. FIPA standard Interaction artifacts are available to the programmer through abstractions to develop FIPA-compliant MAS. Some reusable FIPA implementations are provided by MAS frameworks such as the CNP implementation in JADE. Thus, we reuse JADE API [3] that includes role behaviors for FIPA standard protocols. For example, the CNP-Initiator implements the initiator role in a FIPA-Contract-Net or Iterated-FIPA-Contract-Net, while the CNP-Participant corresponds to the responder role. These two implementations are mapped to the *CNP* feature.

**Organization artifacts:** Some concepts of the *Generic MAS FM* which are linked to the organization such as *Roles* may be implemented by reusable artifacts at the implementation level. For instance, in JADE API, the classes implementing the behaviours can represent roles that are reusable.

### 4.4 Specific MAS Implementation

During the last activity of Domain Engineering, *Specific MAS reusable artifacts* implement *Specific MAS features* by reusing *Generic MAS artifacts*.

Fig. 5 summarizes the four activities of the Domain Engineering. It illustrates a *Contest MAS* reusable artifact which is at the right side bottom of the figure. This Contest artifact implements the *BuyItem_Goal* Contest feature, by reusing the PerformActionGoal() function that is mapped to the *Mono_Goal* feature.

[3] `http://jade.tilab.com/doc/api/`

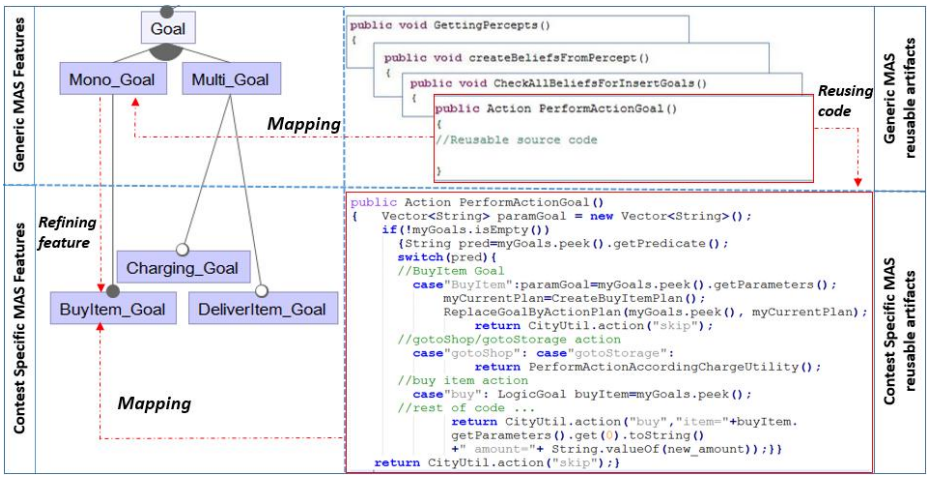


**Fig. 5.** An example of a Contest MAS reusable artifact obtained through the four Domain Engineering activities

## 5 MAS Application Engineering

This section will present application engineering activities, and illustrate them with Contest Agent configuration and derivation.

### 5.1 Requirement Analysis

During this activity, to obtain customized MAS, each requirement is analyzed to detect which features have to be selected from the *Specific MAS FM*, to fill that requirement and constitute a configuration.

Table 1 gives some examples of Contest requirements. The first requirement that corresponds to the Contest Agent Variant CAV1, is fulfilled by the given configuration presented on (a) in Fig. 6. All selected features are represented on (b) through a set of literals. Consequently, the non selected features are excluded from the configuration. The other requirements correspond to Contest Team Variants that consider some organizational aspects. For instance, unlike CTV3, to fill the requirements of CTV1, we must include features such as the *Non_Hierarchical* feature, and exclude others such as the *Hierarchical* one.

### 5.2 Product Derivation

For a valid configuration, the MAS variant derivation activity is automatically achieved by a composer, such as FeatureHouse.

Fig. 6 gives on (c) the derived code relative to the configuration on (a).

For more details, interested readers can access our link [4].

[4] `http://www-desir.lip6.fr/~boufedji/emas17.html`

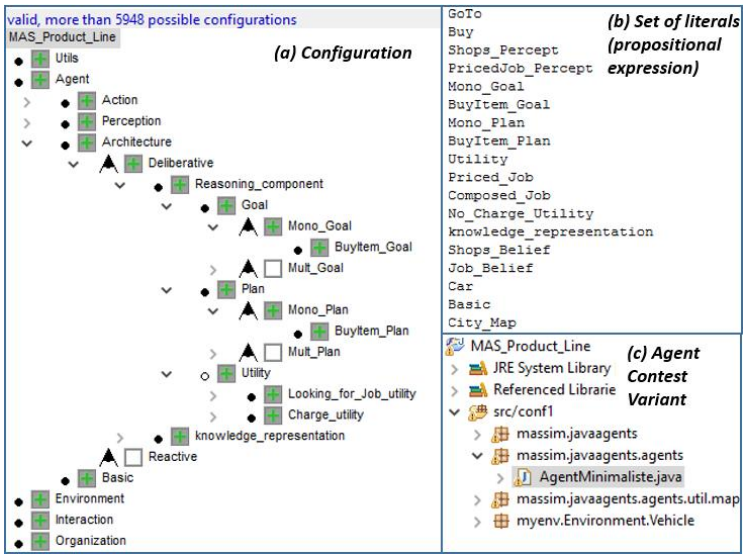


**Fig. 6.** An example of (a) a Contest Agent Configuration (b) its propositional expression and (c) the derived Contest Agent Variant

| Contest Variants | Requirements |
|---|---|
| CAV1 | A Car Contest Agent Variant that has to achieve one goal, by executing an acquisition job (buying items in a shop). The agent has no charge utility concerning its battery, and has to execute one Plan (find a shop, move to it, and buy the items) |
| CTV1 | A Contest Team Variant (CTV) that looks only for priced jobs. The team is organized according to a non hierarchical topology without including groups structures. |
| CTV2 | A Contest Team Variant that looks only for priced jobs. The team is organized according to a non hierarchical topology, with the possibility of structuring into groups of cars and trucks. The groups are supervised by group supervisors. |
| CTV3 | A Contest Team Variant that looks for both priced and auctioned jobs, and that is organized according to hierarchical topology without including groups structures. |
| CTV4 | A Contest Team Variant that looks for both priced and auctioned jobs, and that is organized according to hierarchical topology, with the possibility of structuring into groups of cars and trucks. The groups are supervised by group supervisors. |

**Table 1.** Examples of multi-agent Contest requirements

## 6 Evaluation

*Evaluation objectives:* The main objectives of our evaluation are to show the feasibility of our approach to derive MAS variants and to deduct the rate of reuse improvement our approach brings.

In order to evaluate our approach, we first derived several Contest Agent Variants and deployed them in the MAS Contest environment. Second, we involved groups of students to derive Contest Agent Variants by following the SPLE framework without relying on the generic MAS features and artifacts

that we proposed. After, we compared students' Contest feature models and implementations with ours.

*Contest Agent Variants derivation:*The implementation of our approach adopted the Feature IDE tool[5] to specify the feature models, and to create agent configurations. FeatureHouse composer was used to derive Contest Agent Variants.

The first version of the product line involves neither interaction nor organization aspects of the team, and considers only agents that have to complete their priced jobs. More details are available on the Contest website under the Contest 2016 example [6].

The case study offers more than 5948 possible configurations, as shown on the upper left side in Fig.6. We present among them ten (10) of the derived and simulated Contest Agent Variants. The variants are labeled from CAV1 (Contest Agent Variant) to CAV10. Table 2 represents the relative configurations of each variant that includes both Generic and Contest MAS features labeled respectively GMF and CMF.

CAV1 and CAV2 have the minimal feature configurations that involve a total of six mandatory features. CAV1 corresponds to the simplest agent (see Fig.6). CAV3 to CAV10 represent variants with a maximum number of eight possible selected features. These variants follow the second BDI algorithm [16]; and differ on their charging utilities. According to those variabilities, more or less methods and lines of code are derived. We calculate them by using eclipse Metrics[7].

The percentage of reused features and methods show the two main advantages of our approach. Indeed, the originality of our approach is the possibility to reuse Generic MAS features and artifacts. The percentage of reused features ranges from 25% to 33 %, while the percentage of reused implemented methods ranges from 7% to 10.18 %.

The above results correspond to the rate of reuse improvement our approach offers. The results show also the advantage of using software product lines in our approach. There is a development time saving of about 354 lines of code, and 25 methods from CAV1 to CAV10. We can also compare variants according to their strategies using utilities. Indeed, when using our approach, we can compare Contest agents performances easily since the agents are derived automatically and can be deployed faster. For example, through simulations, we could distinguish between the best and the worst charging utilities. We could also make the agents variants play together.

The second version of the product line involves some interaction and organization aspects of the team. Due to space restrictions we did not include the table. The product line includes four variants labeled from CTV1 (Contest Team Variant) to CTV4. Their configurations correspond to the requirements presented in Table 1. The results show a feature reusing rate that ranges from 27.27 % to 30.76%. These variants which consider organization and interaction

[5] https://marketplace.eclipse.org/content/featureide
[6] http://multiagentcontest.org
[7] http://eclipse-metrics.sourceforge.net

| MAS-PL Features | | | | | | | | | | | | | | | | | | | Metrics | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *GMF* | | | | *CMF* | | | | | | | | | | | | | | | | | | |
| | F1 | F2 | F3 | F4 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | | | | | | |
| **Contest Agent Variants** | Mono_Goal | Multi_Goal | Mono_Plan | Multi_Plan | BuyItem_Goal | Charging_Goal | CallBreakDownService_Goal | Item_Plan | Charging_Plan | CallBreakDownService_Plan | No_Charge_Utility | BreakDown_Utility | ClosestStationFromDestination | ClosestStationFromStartingPosition | ClosestStationWhenHalfEmpty | Car | Truck | Total selected features | Lines of code | Number of methods | % of reused features | % of reused methods |
| CAV1 | x | | x | | x | | | x | | | x | | | | | x | | 6 | 150 | 120 | 33 | 7 |
| CAV2 | x | | x | | x | | | x | | | x | | | | | | x | 6 | 156 | 120 | 33 | 7 |
| CAV3 | | x | | x | x | | x | x | | x | | x | | | | x | | 8 | 152 | 108 | 25 | 10.18 |
| CAV4 | | x | | x | x | | x | x | | x | | x | | | | | x | 8 | 165 | 127 | 25 | 9 |
| CAV5 | | x | | x | x | x | | x | x | | | | x | | | x | | 8 | 176 | 130 | 25 | 8.46 |
| CAV6 | | x | | x | x | x | | x | x | | | | x | | | | x | 8 | 179 | 133 | 25 | 8.27 |
| CAV7 | | x | | x | x | x | | x | x | | | | | x | | x | | 8 | 185 | 133 | 25 | 8.27 |
| CAV8 | | x | | x | x | x | | x | x | | | | | x | | | x | 8 | 182 | 130 | 25 | 8.46 |
| CAV9 | | x | | x | x | x | | x | x | | | | | | x | x | | 8 | 181 | 133 | 25 | 8.27 |
| CAV10 | | x | | x | x | x | | x | x | | | | | | x | | x | 8 | 186 | 133 | 25 | 8.46 |

**Table 2.** Some metrics for ten derived Contest Agents Variants

features presents a higher rate of reused features compared to the agent variants presented above.

*Involving students:* Groups of students were involved in this activity. None of the groups was familiar with SPL concepts. The students were asked to model, implement and derive Contest multi-agent variants by following the SPLE framework that we use in our approach. However, we did not provide them the starting points proposed by our approach, to compare their results with the ones we obtained by relying on both generic MAS FM and artifacts. As a result, we could distinct three categories of groups: CAT1, CAT2 and CAT3.

CAT1 represents groups that failed on detecting domain-independent variability. The category presents the worst results regarding Contest Agent Variants. Indeed, 44.44 % of the students belong to this category.

The students focused only on domain specific variability, what let them build only domain specific features and artifacts.

CAT2 includes groups that detected domain-independent variability, but did not include it in the feature model. It presents a rate of about 33.33 %. This intermediate category succeeded in detecting reusable generic MAS features, but did not exploit them in the feature model. For example, they thought about MAS organizational variabilities such as centralized or decentralized organizations; but did not consider these aspects in possible configurations.

CAT3 concerns groups that detected domain independent variability and introduced it in the feature model. It was the most successful category, but it represents the lowest rate of 22.22 % of the students.

However, the total number of reusable features does not exceed eight, and include *Interaction* and *Organization* features.

The results brings out the advantages of our approach. It can provides CAT1 a starting point to support domain-independent variability. Moreover, it allows CAT2 to exploit the detected features to derive more multi-agent variants. In addition, it provides CAT3 more reusable features and artifacts than those detected. Our approach provides to all categories more possible configurations, which implies more variants.

All these facts lead to increase the number of Contest Agent Variants, save time and effort to the whole categories through the whole process.

Our MAS-PL approach uses of known notations covering both design and implementation aspects, what would facilitate its use and its adoption by MAS developers. Moreover, since it provides a generic MAS FM, MAS designers and developers will not build feature models nor develop the source code from scratch.

However, our approach has some limits. Indeed, it does not consider all organizational artifacts, most variability is specific to contest and the total number of reusable artifacts which should be increased.

But, we prospect to enrich our work with a more refined MAS variability. Currently, we are studying self-organizational aspects of the system. We also prospect to evaluate our approach through other specific domains. We project to evaluate it by students as well.

## 7 Conclusion

In this paper, we proposed a first version of a new MAS-PL approach for the automatic derivation of MAS variants according to MAS requirements. Our MAS-PL approach follows the SPLE framework in both domain and application engineering phases. It relies on two types of features (resp. two types of reusable artifacts): the *Generic MAS* features (resp. artifacts) and the *Specific MAS* features (resp. artifacts). The features of the *Generic MAS feature model* and those of *Specific MAS feature model* are organized according to the Vowels paradigm.

Our MAS-PL approach deals with known notations and covers both the design and implementation aspects. So, it is easy to use by MAS developers. Moreover, it is incremental, the *Specific MAS FM* can be refined as many times as needed to deal with more specific MAS variability.

We illustrated the different activities of our MAS-PL approach by simple examples issued from the Multi-Agent Contest 2016. We derived a Contest product line of agents that includes variants that have been simulated in the Multi-Agent Contest environment. We compared those variants according to some metrics. The result shows that our approach is promising. We also compared these results to those obtained by students without using the Generic MAS features and arti-

facts we proposed. This comparison brought out the value of our Generic MAS features and artifacts in practice.

As for further on-going research work, we are concentrating on an interesting perspective which would allow us to obtain more variability to enrich both *Generic MAS* FM and artifacts. We intend, for instance, to introduce interaction mechanisms such as ant-based algorithms. We also prospect to evaluate our approach by groups of students to compare their results to those presented in this paper. Another perspective is to suggest to researchers and MAS developers to use our approach when implementing Multi-Agent System Product Lines in different domains. The feedback would help us improve our work to serve the technological development and advances.

## References

1. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Springer (2013)
2. Apel, S., Kastner, C., Lengauer, C.: Language-independent and automated software composition: The FeatureHouse experience. Software Engineering, IEEE Transactions on 39(1), 63–79 (2013)
3. Ayala, I., Horcas, J.M., Amor, M., Fuentes, L.: Using models at runtime to adapt self-managed agents for the iot. In: German Conference on Multiagent System Technologies. pp. 155–173. Springer (2016)
4. Behrens, T.: Towards Building Blocks for Agent-Oriented Programming. Ph.D. thesis, Clausthal University of Technology (2012)
5. Behrens, T.M., Hindriks, K.V., Dix, J.: Towards an environment interface standard for agent platforms. Annals of Mathematics and Artificial Intelligence 61(4), 261–295 (2011)
6. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming 78(6), 747–761 (2013)
7. Cernuzzi, L., Juan, T., Sterling, L., Zambonelli, F.: The gaia methodology. In: Methodologies and Software Engineering for Agent Systems, pp. 69–88. Springer (2004)
8. Cirilo, E., Nunes, I., Kulesza, U., Lucena, C.: Automating the product derivation process of multi-agent systems product lines. Journal of Systems and Software 85(2), 258–276 (2012)
9. Cossentino, M.: From requirements to code with the passi methodology. Agent-oriented methodologies 3690, 79–106 (2005)
10. Cossentino, M., Burrafato, P., Lombardo, S., Sabatucci, L.: Introducing pattern reuse in the design of multi-agent systems. In: Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World. pp. 107–120. Springer (2002)
11. Dehlinger, J., Lutz, R.R.: Gaia-PL: A product line engineering approach for efficiently designing multiagent systems. ACM Transactions on Software Engineering and Methodology (TOSEM) 20(4), 17 (2011)
12. Demazeau, Y.: From interactions to collective behaviour in agent-based systems. In: In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo. Citeseer (1995)

13. Ferber, J.: Multi-agent systems: an introduction to distributed artificial intelligence, vol. 1. Addison-Wesley Reading (1999)
14. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Agent-Oriented Software Engineering IV, pp. 214–230. Springer (2004)
15. Ferber, J., Michel, F., Báez, J.: AGRE: Integrating environments with organizations. In: Environments for multi-agent systems, pp. 48–56. Springer (2005)
16. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desire-intention model of agency. In: International Workshop on Agent Theories, Architectures, and Languages. pp. 1–10. Springer (1998)
17. Guessoum, Z., Cossentino, M., Pavón, J.: Roadmap of agent-oriented software engineering. In: Methodologies and software engineering for agent systems, pp. 431–450. Springer (2004)
18. Hara, H., Fujita, S., Sugawara, K.: Reusable software components based on an agent model. In: Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000. pp. 447–452. IEEE (2000)
19. Hübner, J.F., Sichman, J.S., Boissier, O.: Moise+: towards a structural, functional, and deontic model for mas organization. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. pp. 501–502. ACM (2002)
20. Nunes, I., Cowan, D., Cirilo, E., De Lucena, C.J.: A case for new directions in agent-oriented software engineering. In: International Workshop on Agent-Oriented Software Engineering. pp. 37–61. Springer (2010)
21. Nunes, I., De Lucena, C.J., Cowan, D., Kulesza, U., Alencar, P., Nunes, C.: Developing multi-agent system product lines: from requirements to code. International Journal of Agent-Oriented Software Engineering 4(4), 353–389 (2011)
22. Nunes, I., Kulesza, U., Nunes, C., Cirilo, E., Lucena, C.: Extending PASSI to model multi-agent systems product lines. In: Proceedings of the 2009 ACM symposium on Applied Computing. pp. 729–730. ACM (2009)
23. Peña, J., Hinchey, M.G., Ruiz-Cortés, A.: Multi-agent system product lines: challenges and benefits. Communications of the ACM 49(12), 82–84 (2006)
24. Peña, J., Hinchey, M.G., Ruiz-Cortés, A., Trinidad, P.: Building the core architecture of a nasa multiagent system product line. In: Agent-Oriented Software Engineering VII, pp. 208–224. Springer (2007)