An Automated Approach to Manage MAS-Product Line Methods

Sara Casare[1], Tewfik Ziadi[2], Anarosa A. F. Brandão[2,3], and Zahia Guessoum[2,4(✉)]

[1] LTI, Universidade de São Paulo, São Paulo, Brazil
sjcasare@uol.com.br
[2] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, 75005 Paris, France
{tewfik.ziadi,zahia.guessoun}@lip6.fr
[3] Escola Politécnica, Universidade de São Paulo, São Paulo, Brazil
anarosa.brandao@usp.br
[4] CReSTIC, Université de Reims Champagne Ardenne, 51000 Reims, France

Abstract. Multiagent systems (MAS) can vary in several ways: by involving different agents, distinct interaction patterns, various forms of agent organizations and environments. One promising approach to consider this variability in MAS is the use of the concept of Multiagent System-Product Line (MAS-PL). The idea is to implement a family of MAS that belong to the same domain, instead of a single MAS. However, there is still a lack of methodological support to develop MAS-PL. This paper tackles this problem with a rigorous respect of software product line principals. We propose an automated approach, the Meduse for MAS-PL, to generate families of MAS-PL methods that offer software product line best practices integrated with existing MAS development approaches to support MAS-PL development. To illustrate, we present a case study involving a family of MAS-PL methods that extends Gaia and Tropos.

Keywords: Multiagent systems · Software product line · Variability · Method

1 Introduction

Managing variability in Multiagent systems (MAS) has been identified during the last years as one of the main issues within Agent-Oriented Software Engineering (AOSE) [2, 10, 18, 19]. Indeed, the intrinsic properties of MAS, such as modularity, make them variability-rich systems. The notion of software variability is defined as the ability of a software system to be changed, customized or configured for use in a particular context [25]. Therefore, AOSE approaches should consider this variability to design and develop not only a single MAS, but families of systems at the same time.

More than twenty AOSE methods have been proposed by the MAS community to support MAS development [6]. Each of them proposes a set of development activities for analyzing, designing, and implementing the typical MAS components: agents, environment, interaction, and organization [12]. Gaia [26] and Tropos [3] are among the most popular AOSE methods. However, they only proposed to design and develop a

A. El Fallah-Seghrouchni et al. (Eds.): EMAS 2017, LNAI 10738, pp. 180–197, 2018.
https://doi.org/10.1007/978-3-319-91899-0_11

single MAS variant at time. Especially, they do not propose in their initial definition any explicit activity to manage variability and develop families of systems rather than a single system. One promising approach for tackling this limitation has been already identified by the MAS community since the early 2000's through the concept of Multiagent System-Product Line (MAS-PL) [10]. Instead of considering a single MAS, MAS-PL aims to organize a family of multiagent systems according to their similarities (i.e., commonalities) and differences (i.e., variabilities) in order to build MAS customized according to specific needs. MAS-PL reuses concepts proposed by Software Product Line Engineering (SPLE), which is a systematic approach for variability management proposed by the Software Engineering community [1, 8].

Although already introduced into the AOSE community, there is still a lack of methodological support to implement MAS-PL. As we will see in Sect. 2, among the more than twenty AOSE methods, only few of them support the specificities of MAS-PL, e.g. Gaia-PL [11], Peña et al. [19], and Nunes et al. [18]. Nevertheless, in these methods the proposed extensions are very light and do not cover all Software Product Line activities. Moreover, they are tailored to develop MAS families in specific domains and may not be suited for MAS families in diverse domains.

This paper tackles the mentioned issues by proposing **Meduse for MAS-PL**, an automated approach that provides steady methodological support for MAS-PL development by integrating SPLE best practices with several existing AOSE methods. Indeed, this approach results from cooperation between AOSE and SPLE research teams: we capitalize all knowledge on SPLE activities in a way that such activities can be automatically incorporated into AOSE existing methods as a set of SPLE best practices in order to develop MAS families in diverse domains. The proposed approach follows Method Engineering techniques [4, 14] and is built upon the Meduse framework [7]. To illustrate our approach we present a case study that shows how we can provide MAS-PL methods that take into account SPLE best practices to extend two popular AOSE methods, Gaia and Tropos.

The paper is organized as follows: Sect. 2 presents background and motivations. Section 3 presents our approach for dealing with method extensions to develop MAS-PL. Section 4 illustrates such an approach using a case study. Finally, Sect. 5 concludes this work and discusses future work.

2 Background and Motivations

In this section we present some basic notions related to SPLE and MAS-PL methods that are essential for understanding the main aspects of our approach, as well as a motivating example based on Tropos.

2.1 Multiagent System Product Line Methods

MAS-PL methods reuse concepts of SPLE for MAS families' development. Before discussing existing MAS-PL methods, we will briefly introduce the general SPLE framework. Figure 1 shows its four main activities - domain analysis, domain

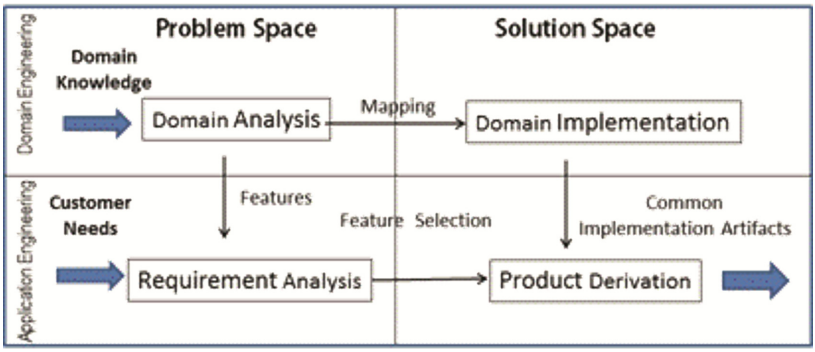

Fig. 1. Software Product Line Engineering general framework [1]

implementation, requirement analysis, and product derivation - organized in the Domain and Application Engineering levels, as well as in the Problem and Solution spaces [1].

Domain Analysis: This activity aims to define the scope of the problem to be tackled and explicitly specify commonality and variability between products that are included in the product line. The results of domain analysis are usually documented in a variability model. Several formalisms have been proposed to specify such a variability model. Among them, we distinguish *Feature Model* [16] and *Decision Model* [23]. The former derives from the work on Feature Oriented Domain Analysis (FODA) [16], while the latter has its roots on the Synthesis method [5, 23].

A feature consists of a distinctive user-visible characteristic of a software product line, used to represent identifiable functional abstractions that shall be present in the final product [16]. It usually encompasses commonalities and variabilities. A feature model describes relationships between features, and formally specifies which feature selections are valid. This is made by hierarchically organizing features in a tree, where edges are used to represent parent-child relationships (see next section, Fig. 3). These parent-child relationships could be of the following types: *mandatory* (the child feature must be present in a product whenever its parent appears); *optional* (the child feature may be present when its parent appears); *alternative* (exactly one child feature must be present when the parent feature appears); *or* (at least one child feature must be present whenever the parent feature appears). Moreover, a set of cross-branch constraints is used to indicate dependencies between features pertaining to different tree branches. Finally, in order to establish their consistency feature models can be described through a set of propositional formulas. Thus, a Satisfiability (SAT) solver tool can be used to determine the feature model consistency, i.e. if it will generate at least one valid product, or whether a given product is valid.

Decision Model is another way of modelling variabilities. In this kind of models, decisions describe the variabilities available in a product line, and specify the set of choices during product derivation. Therefore, taking a decision involves analyzing multiple options and then selecting those that better reflect the customers' needs.

Domain Implementation: In this activity the commonalities and variabilities previously specified are developed as a set of reusable artifacts (also called assets) and organized according to the specified variability model.

Among the domain implementation approaches we can cite Delta-oriented Programming [20, 21], a compositional and modular approach to manage variability based on modifications applied to a core product, which is transformed into another variant of the product line by incrementally applying a set of *delta modules* that propose additions, removals, or alterations of elements. A *condition* is a propositional constraint attached to every delta module through a *when* clause and it determines for which features the specified modifications are to be carried out. Therefore, conditions create the connection between the modifications prescribed in delta modules and the features. A *list of delta modules* and *attached conditions* determines the modifications required to implement different products of the product line, as well as the order in which such modifications shall be applied. Indeed, such a list groups delta modules in ordered partitions and partitions can be partially ordered, i.e., while the order of partitions is fixed, deltas in the same partition can be applied in any order. Finally, the core product can be an empty product.

Requirement Analysis: During this activity needs of a specific customer are considered in order to select the required variabilities (e.g. feature selection), also called a configuration [1]. Therefore, the variability model is instantiated according to the customer requirements.

Product Derivation: Once we have a selection of the required variabilities representing customer needs, the product derivation activity aims to generate (or derive) the product itself. As underlined by Apel et al. [1], depending on the implementation approach this activity can be automated. For compositional approaches as Delta-oriented Programming, the idea is to use what is referred as *composer* to derive product variants.

Extending AOSE methods to support MAS-PL should consider the integration of the different SPLE activities discussed above. As mentioned early, there are just a few MAS-PL methods. Each of them extends particular AOSE method(s) to integrate SPLE activities. However, this integration is often partial and it only concerns some of the SPLE activities previously presented. For instance, Nunes et al. propose a Domain Engineering method to develop MAS-PL that uses PLUS [13] as the SPLE approach, and PASSI [9] combined with MAS-ML [22] as AOSE method.

Peña et al. propose the use of MaCMAS [19] and UML to define the core architecture of a MAS-PL. Their method only considers the Domain Engineering level and it adopts MaCMAS models in several levels of abstraction to guide the building of the MAS-PL variability model. In the same way as Nunes and colleagues, Pena and colleagues' approach results into a single MAS-PL, instead of a family one. Dehlinger and Lutz [10, 11] propose an approach that combines SPLE techniques and Gaia. The method is called Gaia-PL and it is devoted to the Domain Engineering level of a software product line. Their goal was reusing requirements specifications and a heuristic is provided to guide the building of the feature model. Nevertheless, the proposed MAS-PL approaches may not support different application domains than the ones presented as running example.

2.2 Motivating Example

We consider the Tropos method as a running example to discuss the current methodological limitations in the MAS-PL development: MAS-PL methods propose a partial integration of SPLE activities with particular AOSE methods.

Tropos offers a set of development phases to deal with requirements gathering as well as to analyze, design, and code two MAS components: agents and interaction. Figure 2 depicts the five subsequent phases of Tropos, from requirement to implementation phases. We follow the standard notations of the *Software and System Process Engineering Meta-model* (SPEM) [17], which is the *de facto* standard for representing development methods [15].

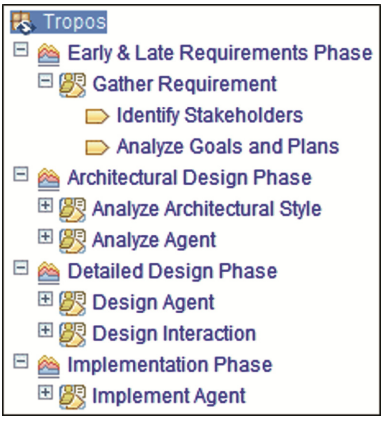

Fig. 2. The Tropos phases using the SPEM notations.

As initially proposed, Tropos only considers the phases that are related to the development of a single MAS without any methodological support for MAS-PL development. To be suitable for MAS-PL development, the original phases of Tropos (see Fig. 2) should be extended by integrating the SPLE activities. For instance, we need to add the domain analysis phase to specify variability. In addition, we also need to include activities that are related to domain implementation and product derivation. Many interesting issues can be considered in this context:

- How can we integrate the different SPLE activities with the Tropos method of Fig. 2, concerning both MAS-PL domain and application engineering levels?
- How can we provide MAS-PL methods based on Tropos for developing MAS families in diverse domains?
- As early discussed in Sect. 2.1, there are two kinds of formalisms to specify the variability model during SPLE domain analysis: Feature Model and Decision Model. Then, how can we manage these different integration possibilities into Tropos?
- How can we automatically generate a new MAS-PL method based on Tropos and SPLE best practices according to project needs?

The next section presents how our approach deals with these issues. As we will show, Meduse for MAS-PL can automatically generate variants of MAS-PL method based on Tropos, as well as based on other AOSE methods like Gaia or PASSI. Moreover, it offers a set of activities based on SPLE best practices that are ready to be fully integrated with these AOSE methods. Finally, our approach provides MAS-PL methods for developing MAS families in diverse domains.

3 From MAS Methods to MAS-PL Methods with Meduse

In this section we present the Meduse for MAS-PL approach. In few words, it is an automated approach to generate methods for developing MAS-Product Lines that provides a steady methodological support following SPLE best practices. Besides, it promotes the reuse of existing AOSE methods, by extending them to explicitly support MAS-PL development according to project needs. Finally, to speed up this extension Meduse for MAS-PL capitalizes all knowledge on SPLE activities proposed in [1], providing SPLE best practices ready to be used for MAS development.

To achieve such a goal our approach takes advantage of Method Engineering techniques. Moreover, it is based on the Meduse framework, which itself adopts SPLE techniques. Therefore, Meduse for MAS-PL uses SPLE principles in two levels. First, SPLE techniques are used in a meta-level fashion to generate method variants to develop MAS-PL. Second, these method variants are used to develop MAS product lines and then to generate MAS applications, i.e. product variants. Since our approach is based on the Meduse framework, we start presenting it.

3.1 Meduse in a Nutshell

Meduse Framework is a general approach to generate software development methods. Meduse adopts SPLE techniques to manage method variability, as well as to automatically derive method variants. Moreover, it adopts Method Engineering principles to manage reusable method artifacts, so-called method fragments, which consist of standardized building blocks based on a coherent part of method [14]. Method variants are built upon these fragments.

Figure 3 presents the big picture of the Meduse framework, showing from the method domain analysis to the final method variant, which is automatically generated according to project needs. First, Meduse proposes to represent method domain knowledge in terms of similarities and differences among variants of a same method family by means of a feature model.

Second, during method domain implementation these method's commonalities and variabilities are developed as a set of method fragments, and organized according to the feature model. In order to do that Meduse proposes reusable method fragments together with a compositional approach based on Delta-oriented programming (see Sect. 2). Thus, method derivation relies on the application of delta modules over an empty method: the modification proposed by a delta module consists of the additions and/or removals of reusable method fragments from the method variant. Therefore, one can

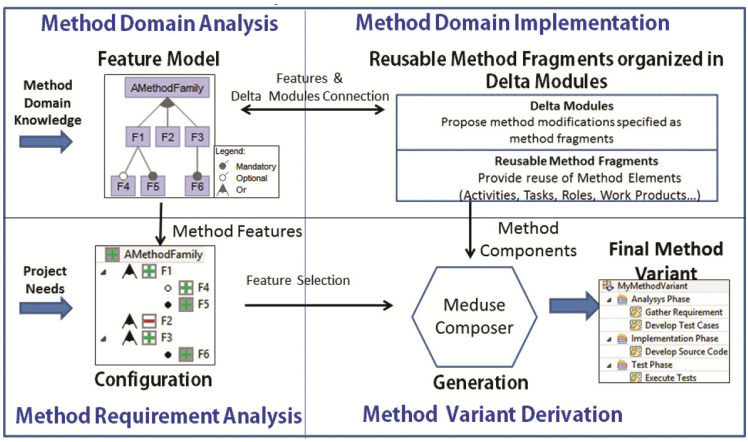

Fig. 3. A big picture of Meduse framework

define the partially-ordered sequence in which method fragments will appear in the method variant, since delta modules are grouped in fixed-ordered partitions, and modules in the same partition can be applied in any order.

Third, method variants are specified through a method configuration, which consists of a set of features selected according to project needs. Finally, Meduse proposes a tool – the Meduse Composer - that automatically derives the specified method variant: it takes the empty method as starting point of the work breakdown structure and incrementally adds or removes method fragments according to the modifications specified in the delta modules connected with the selected features through attached conditions. Figure 3 depicts the final method variant automatically generated by the Method Variant Derivation activity.

3.2 Managing MAS-PL Method Family with Meduse

After introducing the Meduse framework we present the Meduse for MAS-PL approach in details.

Figure 4 illustrates how we can implement a family of MAS-PL methods and automatically generate method variants. As previously mentioned, our approach applies SPLE principles in two levels: to develop a family of MAS-PL methods and then to develop MAS product lines themselves. First, a method family for MAS-PL is specified in terms of commonalities and variabilities among methods. Second, method family implementation is speeded up with a set of reusable method fragments offered by the Meduse for MAS-PL approach. Such fragments concern SPLE best practices and are ready to be integrated to AOSE methods. Third, a particular method for developing MAS product lines is configured over the method's commonalities and variabilities available in the method family, and then this final method is automatically generated and used to support the development of MAS product lines.

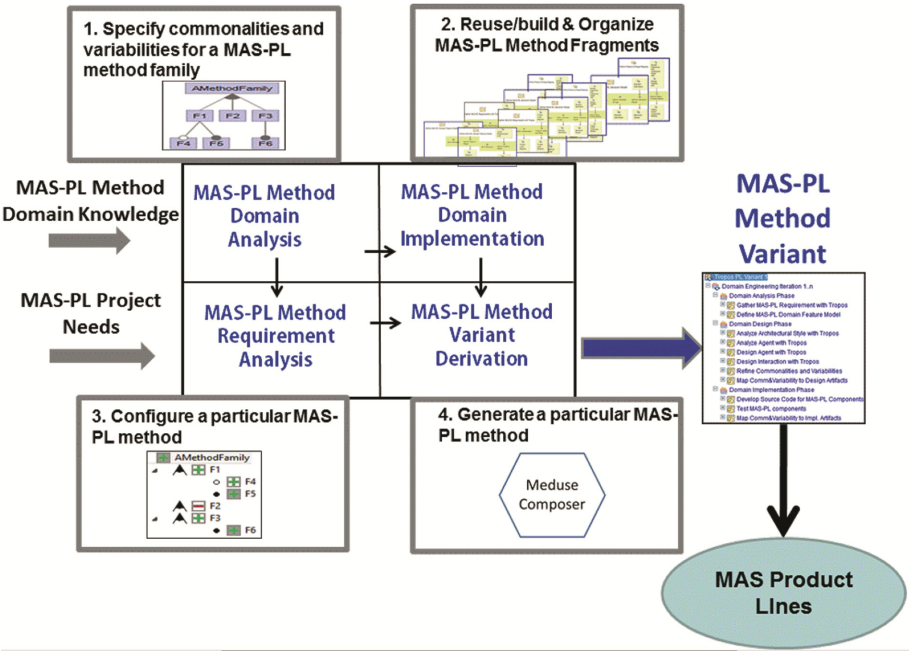

Fig. 4. Generating method variants for developing MAS product lines

Specifying Commonalities and Variabilities for a Family of MAS-PL Methods MAS-PL method variants are generated through method families where the MAS-PL methodological knowledge is represented as method commonalities and variabilities by means of a feature model. Thus, on one hand features represent the various AOSE methods that may be extended, like Tropos, Gaia, or PASSI. On the other hand, features represent a large set of SPLE activities and related techniques that may be chosen as best practices to develop MAS-PL. Examples of such activities are those described by the general SPLE framework proposed in [1] (see Sect. 2). One of the main advantages of using feature models to specify a MAS-PL method family is that we can use SAT solver tools to determine whether such a feature model is consistent, i.e. if at least one valid method configuration exists, and whether or not a given configuration is valid. Therefore, a valid MAS-PL method variant is guaranteed.

Reusing/Building and Organizing Method Fragments for MAS-PL Method Family The first step of the MAS-PL Domain Implementation consists of reusing and/or building a set of method fragments that implements the commonalities and variabilities encompassed by a family of MAS-PL methods. In order to speed up this implementation, we have capitalized all knowledge on SPLE activities proposed in [1] as a set of reusable method fragments. Therefore, Meduse for MAS-PL offers several reusable fragments that provide detailed guidance to develop MAS-PL based on SPLE best practices. These fragments were sourced from FODA, Synthesis, and the Apel et al. approach, and are ready to be integrated to existing AOSE methods. For instance, some of these reusable

fragments deal with feature and decision models, while others deal with the analysis of a particular MAS application over a MAS product line, and the generation of the final code of this MAS application. All of these method fragments encompass fine-grained elements, like tasks, work products, and roles. Figure 5 illustrates four of them: *Define MAS-PL Decision Model, Define MAS-PL Feature Model, Perform MAS Application Requirement Analysis, and Generate MAS Product Code*. For instance, the first is sourced from Synthesis and encompasses two tasks: *Define Domain Scope* and *Define Decision Model*. Such tasks are performed by the *System Analyst* role and produce *Domain Definition* and *Decision Model* as work products, respectively.

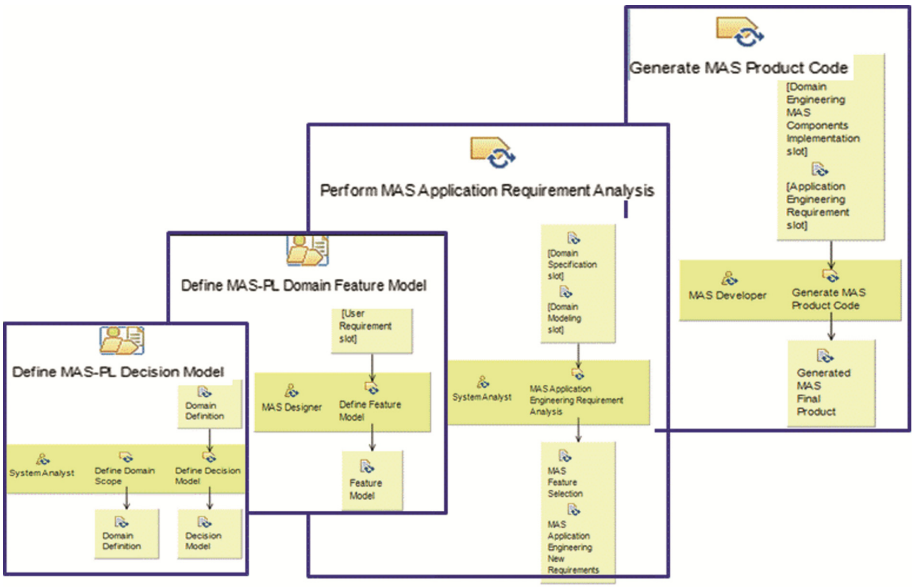

Fig. 5. Example of four method fragments provided by Meduse for MAS-PL

On the other hand, fragments related to AOSE may be provided by existing method fragment's libraries, like the Medee framework [6] that offers more than a hundred fragments sourced from popular AOSE methods. Moreover, method fragments may be also built from scratch whenever the method domain analysis identifies a new feature not yet implemented as reusable asset, related both to AOSE or SPLE.

The second step of the MAS-PL Domain Implementation consists of specifying delta modules that describe the modification to be applied to a method variant by incrementally adding or removing method fragments. These delta modules are then associated with the conditions in which they should be applied. As explained in Sect. 2, these conditions create the connection between the modifications prescribed in delta modules and the feature model. Finally, delta modules are put together in an ordered list and then are ready to be applied during the generation of a particular MAS-PL method variant. Nevertheless, before such a generation we have to configure this method.

Configuring a Particular MAS-PL Method
To configure a particular MAS-PL method we should analyze the project at hand needs and express them through a selection of the AOSE and SPLE development activities among those provided by the MAS-PL method family. In order to result in a valid method configuration such a selection must take into account the relationships specified in the feature model. As previously explained, Meduse for MAS-PL proposes the use of SAT solver tools to determine whether a given configuration is valid.

For instance, we may select Tropos as the AOSE method to be extended by integrating SPLE activities to guide the analysis, design, and implementation of a MAS product line concerning the development of several agents, their roles and interactions. Moreover, these SPLE activities may also guide the requirement analysis and derivation of the final MAS application.

Generating a MAS-PL Method Variant
Finally, the final MAS-PL method is automatically generated by the Meduse Composer and can be used to support the development of MAS product lines.

The following steps are performed by the Meduse Composer tool. First, it finds all delta modules that shall be applied to the MAS-PL method variant, i.e., those delta modules attached to a condition evaluated to true for the configuration. For instance, a condition defined as the propositional formula *Tropos and Decision Model* is evaluated to true whenever a method configuration includes these two features, and therefore all delta modules attached to this condition are taken into account during method derivation. Second, that tool generates an empty MAS-PL method variant and then adds and/or removes reusable method fragments from that variant according to the modification proposed by the selected delta modules, always respecting the order defined by the list of modules.

We have developed a prototype implementation of Meduse for MAS-PL, in which we adopted FeatureIDE [24] as tool to specify the feature model and to create method configurations, as well as SPEM and Eclipse Process Framework (EPF)[1] to manage method fragments, delta modules, and the derived method variants. Interested readers may access the Meduse for MAS-PL website[2] to see available reusable method fragments sourced from several SPLE approaches, and method variants for MAS-PL development automatically derived during the case study presented in the next section.

4 Creating MAS-PL Method Family: A Case Study

In this section we present a case study that shows how we can use our approach to extend two AOSE methods - Gaia and Tropos - to support MAS-PL development following SPLE best practices. It consists of creating a MAS-PL method family and then deriving several method variants.

The scope of this method family is defined as follows. It aims at providing methods to develop MAS product lines based on Gaia or Tropos. Moreover, such method family

[1] https://eclipse.org/epf/.
[2] https://pages.lip6.fr/Tewfik.Ziadi/EMAS17/.

would offer two techniques for modeling MAS variabilities - feature model and decision model - as well as it would deal with the development of four MAS components: Agent, Environment, Interaction, and Organization. Finally, this method family would involve both Domain and Application Engineering levels to provide a full SPLE development cycle: from MAS-PL Domain Analysis to the MAS-PL Product Derivation. It should be noted that another scope definition, for instance involving different MAS components or other AOSE methods, like PASSI, is also possible and would give rise to a different MAS-PL method family.

The resulting feature model contains eighteen features, where eleven of them represent the similarities among the method variants pertaining to such a family, while the remainder seven features represent how such method variants may vary. To implement the method family we reused a set of method fragments sourced from Gaia and Tropos, as well as those provided by the Meduse for MAS-PL approach concerning FODA, Synthesis, and Apel et al. Finally, the Meduse Composer was used to derive the twenty possible variants. The remainder of this section describes this case study in detail.

4.1 Feature Model for a MAS-PL Method Family

As aforementioned, similarities and differences among the method variants of our MAS-PL method family were specified through the feature model presented in Fig. 6. This feature model is composed of eighteen features and one constraint (environment v organization => Gaia). The root of that diagram is labelled with *MAS_PL_Method* to represent a MAS-PL development method. It has three mandatory child features: *MASComponent, MASMethod*, and *SPLEngineering*.

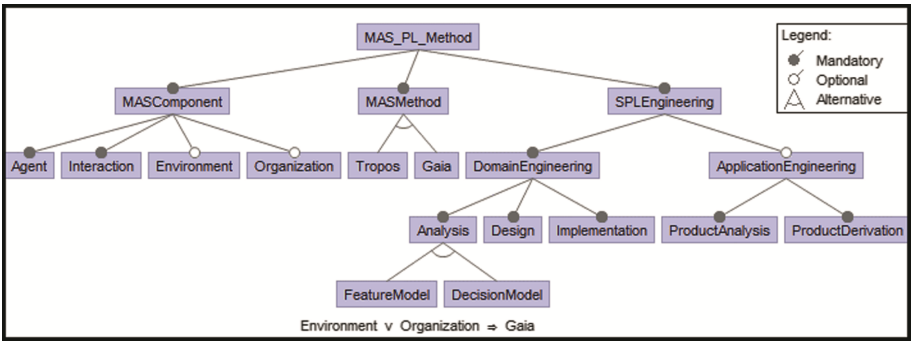

Fig. 6. A feature model diagram for a MAS-PL method family

MASComponent has two mandatory and two optional child features – *Agent* and *Interaction*, *Environment* and *Organization* - respectively, because in this case study we consider that a MAS-PL method must cover at least the development of agents and their interaction, and may cover also environment and organization development. Thanks to the unique constraint defined in this model, whenever a method variant includes the features *Environment* or *Organization* it must include *Gaia* as MASMethod, because

Gaia deals with the development of the four MAS components, while *Tropos*, the optional MAS-Method child feature, takes into account only the development of Agent and Interaction.

Moreover, *SPLEngineering* has the two SPLE levels as child features: *DomainEngineering* and *ApplicationEngineering*. The former is a mandatory feature while the latter is an optional one because in this case study we consider that a MAS-PL method must propose at least the Domain Engineering level. The *DomainEngineering* feature has three mandatory child features – *Analysis, Design, Implementation* – which correspond to the three domain development phases proposed by Apel et al. Additionally, *Analysis* offers two alternative child features - *FeatureModel* and *DecisionModel* - that represent the choice of SPLE techniques to model MAS variability. Therefore, exactly one of these techniques must be present in a MAS-PL method variant.

Finally, whenever a MAS-PL method includes the *ApplicationEngineering* feature it must include its two child features: *ProductAnalysis* and *ProductDerivation*. It should be observed that such a feature model corresponds to the previously described scope. A diverse scope would give rise to a different feature model.

4.2 Domain Implementation for a MAS-PL Method Family

While similarities and differences among the method variants of the MAS-PL method family were specified through a feature model, we used reusable fragments and Delta-oriented programming to implement such features. Therefore, the implementation of that MAS-PL method family comprised a set of reusable method fragments organized in a list of delta modules. Method fragments were sourced from the two AOSE methods considered in our scope, Gaia and Tropos. Concerning SPLE approaches, we have reused those reusable fragments provided by the Meduse for MAS-PL approach.

The rest of this section describes how we reused method fragments and implemented delta modules, starting with method fragments.

Method Fragments for MAS-PL Family

In order to deal with SPLE activities and techniques we reused ten of the set of fragments provided by Meduse for MAS-PL. These fragments were sourced from FODA (for feature model), Synthesis (for decision model), and Apel et al. (for SPLE domain and application activities). Besides, method fragments sourced from Gaia and Tropos were provided by the Medee Framework: (i) six fragments sourced from Gaia were used to guide the analysis and design of agents, environment, interaction, and organizations; (ii) five fragments sourced from Tropos were used to analyze and design agents and interaction. Summing up, this case study involved twenty-one method fragments.

Figure 7 shows two examples of these method fragments, one sourced from Tropos (*Gather MAS-PL Requirement with Tropos*) and other sourced from Synthesis (*Define MAS-PL Decision Model*).

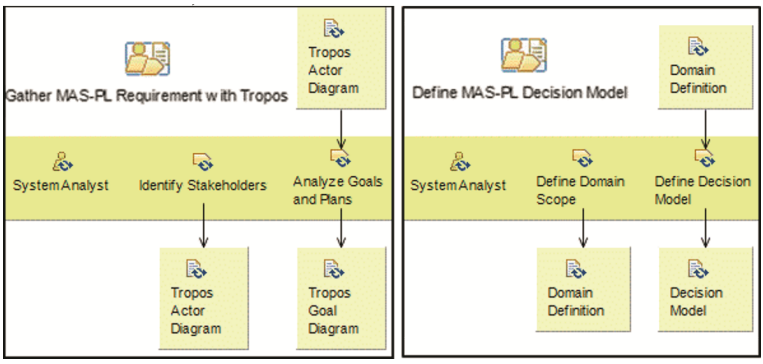

Fig. 7. Examples of two method fragments for a MAS-PL method family, sourced from Tropos (left) and Synthesis (right).

Delta Modules for MAS-PL Method Family

A list of eleven delta modules were implemented to specify the modifications to be applied during the derivation of MAS-PL method variants according to a given method configuration.

Figure 8 (left) illustrates four of these delta modules, *D-TroposFMAnalysis, D-GaiaDMAnalysis, D-TroposDMAnalysis, and D-GaiaFMAnalysis*, as well as the comprised method fragments of the first two delta modules. Note that two of these fragments correspond to those depicted in Fig. 7, namely *Gather MAS-PL Requirement with Tropos* and *Define MAS-PL Decision Model* fragments.

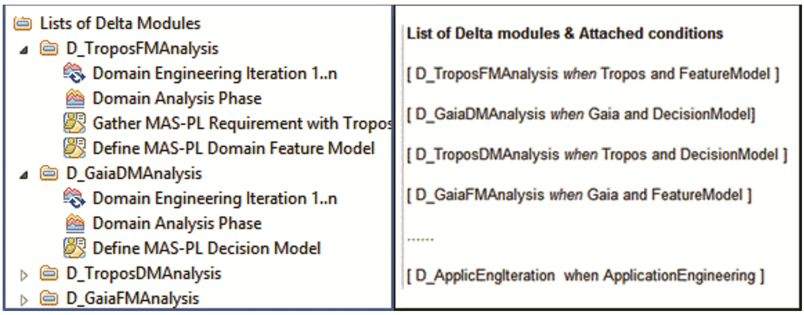

Fig. 8. A partial representation of the List of Delta modules containing method fragments (left), and the list of Delta modules and attached conditions (right)

Moreover, the list of delta modules determines the order in which such delta modules must be applied during method derivation (see Fig. 8 right). As explained before, delta modules were attached to conditions which allow their connection with combinations of features. For instance, a MAS-PL method variant would include the four method fragments defined in the *D-TroposFMAnalysis* delta module whenever the *Tropos* and *FeatureModel* features were selected to configure the MAS-PL method variant.

Method Configuration and Derivation of MAS-PL Method Variants

After implementation, the MAS-PL method family is ready to give rise to method variants according to a given method configuration. As shown in Table 1 (left), the eighteen features proposed in this case study offered twenty possible MAS-PL method configurations, four of them based on Tropos and sixteen based on Gaia.

For instance, Configuration 1 is among the smallest MAS-PL method configurations and encompasses eleven features: MAS component, Agent, Iteration, MAS Method, Tropos, SPL Engineering, Domain Engineering, Analysis, Feature Model, Design, Implementation. On the other hand, Configurations 19 and 20 encompass sixteen features and therefore are among the largest ones. The remaining possible configurations encompass sets ranging from twelve to fifteen features.

Table 1. MAS-PL method configurations and derived MAS-PL method variants.

MAS-PL Method Configuration																			MAS-PL Method Variants	
	MAS-PL Features																			
Configuration #	F1 MAS Component	F2 Agent	F3 Environment	F4 Interaction	F5 Organization	F6 MAS Method	F7 Gaia	F8 Tropos	F9 SPL	F10 Domain Engineer.	F11 Analysis	F12 Feature Model	F13 Decision Model	F14 Design	F15 Implementation	F16 Applic.	F17 Product Analysis	F18 Product Derivation	num. of Selected Features		
1	*	*	_	*	_	*	_	x	*	*	*	x	_	*	*				11	Tropos-PL variants	V1: Domain Eng. w/ FM & A.I.
2	*	*	_	*	_	*	_	x	*	*	*	_	x	*	*				11		V2: Domain Eng. w/ DM & A.I.
3	*	*	_	*	_	*	_	x	*	*	*	x	_	*	*	x	x	x	14		V3: Domain&Application Eng. w/ FM & A.I.
4	*	*	_	*	_	*	_	x	*	*	*	_	x	*	*	x	x	x	14		V4: Domain&Application Eng. w/ DM & A.I.
5	*	*		*		*	x	_	*	*	*	x	_	*	*				11	Gaia-PL Variants	V1: Domain Eng. w/ FM & A.I.
6	*	*		*		*	x	_	*	*	*	_	x	*	*				11		V2: Domain Eng. w/ DM & A.I.
7	*	*	x	*		*	x	_	*	*	*	x	_	*	*				12		V3: Domain Eng. w/ FM & A.E.I.
8	*	*	x	*		*	x	_	*	*	*	_	x	*	*				12		V4: Domain Eng. w/ DM & A.E.I.
9	*	*		*	x	*	x	_	*	*	*	x	_	*	*				12		V5: Domain Eng. w/ FM & A.I.O.
10	*	*		*	x	*	x	_	*	*	*	_	x	*	*				12		V6: Domain Eng. w/ DM & A.I.O.
11	*	*	x	*	x	*	x	_	*	*	*	x	_	*	*				13		V7: Domain Eng. w/ FM & A.E.I.O.
12	*	*	x	*	x	*	x	_	*	*	*	_	x	*	*				13		V8: Domain Eng. w/ DM & A.E.I.O.
13	*	*		*		*	x	_	*	*	*	x	_	*	*	x	x	x	14		V9: Domain&Application Eng. w/ FM & A.I.
14	*	*		*		*	x	_	*	*	*	_	x	*	*	x	x	x	14		V10: Domain&Application Eng. w/ DM & A.I.
15	*	*	x	*		*	x	_	*	*	*	x	_	*	*	x	x	x	15		V11: Domain&Application Eng. w/ FM & A.E.I.
16	*	*	x	*		*	x	_	*	*	*	_	x	*	*	x	x	x	15		V12: Domain&Application Eng. w/ DM & A.E.I.
17	*	*		*	x	*	x	_	*	*	*	x	_	*	*	x	x	x	15		V13: Domain&Application Eng. w/ FM & A.I.O.
18	*	*		*	x	*	x	_	*	*	*	_	x	*	*	x	x	x	15		V14: Domain&Application Eng. w/ DM & A.I.O.
19	*	*	x	*	x	*	x	_	*	*	*	x	_	*	*	x	x	x	16		V15: Domain&Application Eng. w/ FM & A.I.E.O.
20	*	*	x	*	x	*	x	_	*	*	*	_	x	*	*	x	x	x	16		V16: Domain&Application Eng. w/ DM & A.I.E.O.

FM = Feature Model DM = Decision Model A = Agent I = Interaction E = Environment O = Organization

(*) mandatory feature (x) optional selected featured (-) optional disable feature ( ) optional unselected feature

The derivation of a MAS-PL method variant consists of incrementally applying to an empty method the modifications specified by the delta modules attached to valid conditions in a given configuration. Such a derivation was automatically achieved by the Meduse Composer as follows:

(i) Finding all delta modules that shall be applied to the MAS-PL method variant, i.e. those modules attached to a condition evaluated to true for the given configuration. For example, for *Configuration 1* the delta module *D-TroposFMAnalysis* (see

Fig. 8 right) shall be selected, since its attached condition (*Tropos and FeatureModel*) is evaluated to true.

(ii) Generating the method variant by applying the modification proposed by the selected delta modules respecting the order defined by the list of delta modules.

All the twenty MAS-PL method variants presented in Table 1 (right) have been derived and are available in the Meduse for MAS-PL website. Two of these method variants are depicted in Fig. 9: *Tropos-PL Variant 1* and *Gaia-PL Variant 16*. The similarities among these variants are highlighted in red: the two method fragments belonging to Domain Design phase and the entire Domain Implementation phase.

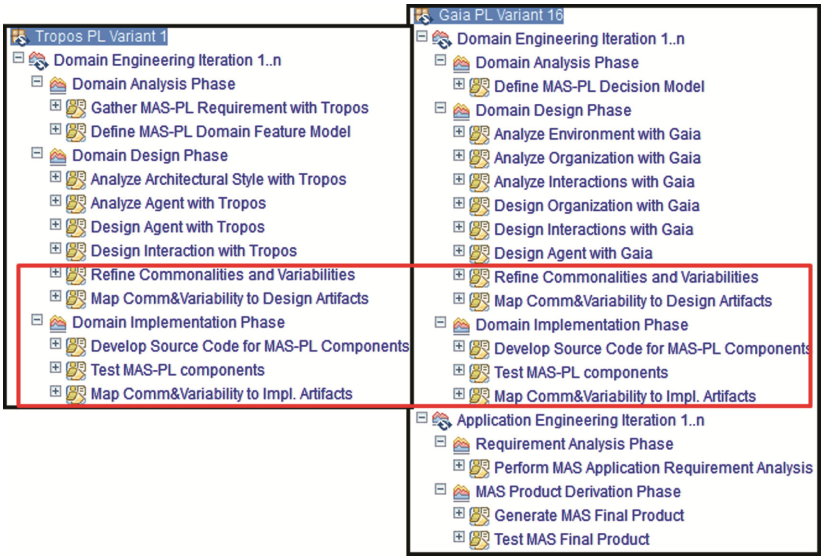

Fig. 9. Two MAS-PL method variants based on Tropos (left) and Gaia (right), highlighting in red the common fragments among them. (Color figure online)

Tropos-PL Variant 1 is among the smallest variants: it encompasses only the Domain Engineering Iteration, which is composed of three phases and eleven activities. It proposes modeling MAS-PL variabilities using a feature model during Domain Analysis Phase. Moreover, it deals with agent and interaction development during Domain Design Phase. On the other hand, *Gaia-PL Variant 16* is among the largest variants: it covers both the Domain and Application Engineering Iterations and includes the four MAS components, i.e. agent, environment, interaction, and organization. It proposes modeling MAS-PL variabilities using a Decision Model, and encompasses five phases that, in their turn, are composed of sixteen activities.

It should be observed that our approach can generate several extensions for the same AOSE method. For instance, the SPLE domain analysis may vary according to the formalism adopted to specify the variability model: feature model or decision model. Therefore, Fig. 10 illustrates two Tropos based MAS-PL method variants generated by

our approach. In addition to the *Tropos-PL Variant 1*, we also generated the *Tropos-PL Variant 4* that uses the decision model instead of feature model and also covers all activities of the SPLE application engineering level.

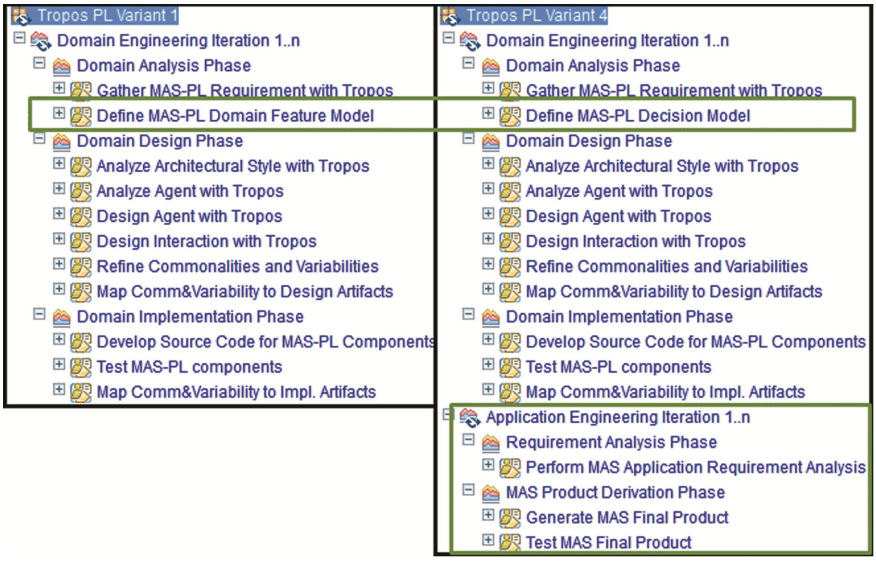

Fig. 10. Two MAS-PL method variants based on Tropos – the smallest (left) and the largest (right) - highlighting in green the variability among them. (Color figure online)

5 Conclusion

This paper introduced Meduse for MAS-PL, an automated approach that aims at providing steady methodological support for MAS-PL development. This approach results from a close collaboration between MAS and SPLE researchers and applies SPLE principles in two levels: to develop a family of MAS-PL methods and then to develop MAS product lines.

Meduse for MAS-PL automatically generates methods for MAS-PL based on existing AOSE methods, by extending them to explicitly support MAS-PL development according to project needs. Moreover, to speed up this extension Meduse for MAS-PL capitalizes all knowledge on SPLE activities proposed in [1], providing SPLE best practices ready to be used for MAS-PL development. Therefore, it promotes the reuse of both AOSE methods and SPLE best practices. Indeed, and as illustrated by the presented case study, our approach provides a set of reusable method fragments sourced from several SPLE development approaches that are ready to be automatically incorporated to AOSE methods.

We validate our approach by the generation of twenty MAS-PL methods that extend Gaia and Tropos. However, its principals can be applied to all AOSE methods. Some of the method variants generated during the case study were used by Master students in

the University Pierre et Marie Curie (Paris 6). Those students apply these MAS-PL methods to support the development of teams to the multiagent contest[3]. They consider the use of the proposed MAS-PL methods to generate several multiagent teams. This experience shows that the generated methods are promising. We plan to propose to several groups of students another project and some criteria to measure the quality of those methods.

Acknowledgement. Sara Casare was supported by CNPq (grant #233828/2014-1), Brazil.

References

1. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Springer, Berlin (2013)
2. Brandao, A., Boufedji, D., Ziadi, T., Guessoum, Z.: Vers une approche d'ingénierie multiagent à base de ligne de produits logiciels. In: 23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2015), Cépaduès, pp. 49–58 (2015)
3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: an agent-oriented software development methodology. J. Auton. Agents Multi-Agent Syst. **8**(3), 203–236 (2004)
4. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Inf. Softw. Technol. **38**(4), 275–280 (1996)
5. Campbell Jr., G.H., Faulk, S.R., Weiss, D.M.: An introduction to Synthesis. Software Productivity Consortium, Herndon (1990)
6. Casare, S., Brandão, A.A., Guessoum, Z., Sichman, J.S.: Medee Method Framework: a situational approach for organization-centered MAS. Auton. Agents Multi-Agent Syst. **28**(3), 430–473 (2014)
7. Casare, S., Ziadi, T., Brandão, A.A., Guessoum, Z.: Meduse: an approach for tailoring software development process. In: Proceedings of the 21st International Conference on Engineering of Complex Computer Systems, ICECCS 2016, pp. 197–200 (2016)
8. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
9. Cossentino, M.: From requirements to code with the PASSI methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-oriented Methodologies, pp. 79–106, Idea Group Inc., Hershey (2005)
10. Dehlinger, J., Lutz, R.R.: A product-line requirements approach to safe reuse in multiagent systems. In: International Workshop on Software Engineering for Large-scale Multi-agent Systems, pp. 1–7 (2005)
11. Dehlinger, J., Lutz, R.R.: Gaia-PL: a product line engineering approach for efficiently designing multiagent systems. ACM Trans. Softw. Eng. Methodol. (TOSEM) **20**(4), 17 (2011)
12. Demazeau, Y.: From interactions to collective behavior in agent-based systems. In: Proceedings of the First European Conference on Cognitive Science. Saint-Malo, pp. 117–132 (1995)
13. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley, USA (2004)

[3] https://multiagentcontest.org/.

14. Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young (1997)
15. Henderson-Sellers, B., Ralyté, J.: Situational method engineering: state-of-the-art review. J. UCS **16**(3), 424–478 (2010)
16. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study (No. CMU/SEI-90-TR-21). Carnegie-Mellon University Pittsburgh Pa Software Engineering Inst. (1990)
17. OMG: Object Management Group. Software & Systems Process Engineering Meta-Model Specification, version 2.0. OMG document number: formal/2008-04-01 (2008). http://www.omg.org/spec/SPEM/2.0/PDF
18. Nunes, I., Lucena, C.J.P., Cowan, D., Kulesza, U., Alencar, P., Nunes, C.: Developing multi-agent system product lines: from requirements to code. Int. J. Agent-Oriented Softw. Eng. **4**(4), 353–389 (2011)
19. Peña, J., Hinchey, M.G., Resinas, M., Sterritt, R., Rash, J.L.: Designing and managing evolving systems using a MAS product line approach. Sci. Comput. Program. **66**(1), 71–86 (2007)
20. Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N.: Delta-oriented programming of software product lines. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 77–91. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15579-6_6
21. Schaefer, I., Damiani, F.: Pure delta-oriented programming. In: Proceedings of the 2nd International Workshop on Feature-Oriented Software Development, pp. 49–56. ACM (2011)
22. Silva, V., Choren, R., Lucena, C.: MAS-ML: A Multi-Agent System Modelling Language. Int. J. Agent-Oriented Softw. Eng. **2**(4), 382–421 (2008)
23. SPC: Software Productivity Consortium Services Corporation. Technical report SPC-92019-CMC. Reuse-Driven Software Processes Guidebook, Version 02.00.03 (1993)
24. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: an extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014)
25. Van Gurp, J., Bosch, J., Svahnberg, M.: On the notion of variability in software product lines. In: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001). IEEE Computer Society, Washington, DC (2001)
26. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the Gaia methodology. ACM Trans. Softw. Eng. Method **12**(3), 317–370 (2003)