Building the Core Architecture of a NASA Multiagent System Product Line*

Joaquin Peña[1], Michael G. Hinchey[2], Antonio Ruiz-Cortés[1], and Pablo Trinidad[1]

[1] University of Seville, Spain
{joaquinp, aruiz}@us.es, trinidad@lsi.us.es
[2] NASA Goddard Space Flight Center, USA
Michael.G.Hinchey@nasa.gov

Abstract. The field of Software Product Lines (SPL) emphasizes building a family of software products from which concrete products can be derived rapidly. This helps to reduce time-to-market, costs, etc., and can result in improved software quality and safety. Current Agent-Oriented Software Engineering (AOSE) methodologies are concerned with developing a single Multiagent System. The main contribution of this paper is a proposal to developing the core architecture of a Multiagent Systems Product Line (MAS-PL), exemplifying our approach with reference to a concept NASA mission based on multiagent technology.

1 Introduction

Many organizations, and software companies in particular, develop a range of products over periods of time that exhibit many of the same properties and features. The multiagent systems community exhibits similar trends. However, the community has not as yet developed the infrastructure to develop a core multiagent system (hereafter, MAS) from which concrete (substantially similar) products can be derived.

The software product line paradigm (hereafter, SPL) augurs the potential of developing a set of core assets for a family of products from which customized products can be rapidly generated, reducing time-to-market, costs, etc. [3], while simultaneously improving quality, by making greater effort in design, implementation and test more financially viable, as this effort can be amortized over several products. The feasibility of building MASs product lines is presented in [16], but no specific methodology is proposed. In this paper, we propose an approach for performing the first stages in the lifecycle of building a multiagent system product line (MAS-PL).

* The work reported in this article was supported by the Spanish Ministry of Science and Technology under grants TIC2003-02737-C02-01 and TIN2006-00472 and by the NASA Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, USA.

For enabling a product line, one of the important activities to be performed is to identify a core architecture for the family of software products. Unfortunately, there is no AOSE methodology that demonstrates how to do this for MAS-PLs. Our approach is based on the Methodology for analysing Complex Multiagent Systems (MaCMAS) [18], an AOSE methodology focused on dealing with complexity, which uses UML as a modeling language and builds on our current research and development experience in the field of SPLs.

Roughly, our approach consists of using goal-oriented requirement documents, role models, and traceability diagrams in order to build a first model of the system, and later use information on variability and commonalities throughout the products to propose a transformation of the former models that represent the core architecture of the family.

The main contributions of this paper are: (i) we introduce feature models in the agent field in order to document variabilities and commonalities across products; (ii) we provide an automatic algorithm and a prototype for performing commonality analysis (that is to say, to automatically analyze the probability that a feature appears in a product); (iii) we propose an operation to compose the models corresponding to a feature that allows us to build the core architecture which includes those features whose probability of appearing is above a given threshold.

2 Motivating MAS-PL with a NASA case study

There has been significant NASA research on the subject of agent technology, with a view to greater exploitation of such technologies in future missions.

The ANTS (Autonomous Nano Technology Swarm) concept mission,[1] for example, will be based on a grouping of agents that work jointly and autonomously to achieve mission goals, analogous to a swarm in nature.

Lander Amorphous Rover Antenna (LARA) is a sub-mission, envisaged for the 2015-2020 timeframe, that will use a highly reconfigurable-in-form rover artifact. Tens of these rovers, behaving as a swarm, will be used to explore the Lunar and Martian surfaces. Each of these "vehicles" or rovers will have the ability to change its form from a snake-like form, to a cylinder, or to an antenna, which will provide them with a wide range of functional possibilities. They are envisaged as possible building materials for future human lunar bases.

Prospecting Asteroid Mission (PAM) is a concept sub-mission based on the ANTS concepts that will be dedicated to exploring the asteroid belt. A thousand pico-spacecraft (less than 1kg each) may be launched from a point in space forming sub-swarms, and deployed to study asteroids of interest in the asteroid belt. Saturn Autonomous Ring Array (SARA) is also a concept sub-mission similar to PAM but whose goal is analysis of the Rings of Saturn.

Although based on mainly the same concepts, these sub-missions differ. For example, in PAM, spacecraft should be able to protect themselves from solar

[1] http://ants.gsfc.nasa.gov/

storms, while in SARA this is not of concern, but as a higher gravitational force exists, the spacecraft should be capable of avoiding gravitational "pull" and collisions with particles of the rings, as well as with other spacecraft. Another example is the mechanism used for motion in these missions. Some of them require ground-based motion, i.e. LARA, while other missions involve flying spacecraft employing gas propulsion and solar sails for power.

Thus, ANTS represents a number of sub-missions, each with common features, but with a wide range of applicability, and hence several products.

Being able to build a MAS-PL for these sets of sub-missions, with a set of reusable assets at all the levels (software artifacts, software processes, engineering knowledge, best practices, etc.), can drastically reduce temporal and monetary costs in the development of such missions.

In [16], a number of challenges are presented in the context of MAS-PL. In this paper we cover some of these challenges, which has motivated this research to address the following issues:

SPL for distributed systems. Distributed systems have not been a hot topic in the SPL field. We will explore a case study based on the ANTS concept mission presented above, which is a highly complex distributed system. Thus, this represents a first step towards addressing this challenge.

AOSE deficiencies. AOSE does not cover many of the activities of SPL. These are mainly concentrated on commonality analysis, and its implications for the entire SPL approach. This motivates us to cover this issue, validating our approach with the case study presented.

3 Background information

As a result of combining two different fields, we have to contextualize our work in both research areas. In this section, we provide an overview of SPL and AOSE illustrating the points of synergy between them.

3.1 Software Product Lines

The field of software product lines covers the entire software lifecycle needed to develop a family of products where the derivation of concrete products is achieved systematically or even automatically when possible.

Its software process is usually divided into two main stages: *Domain Engineering* and *Application Engineering*. The former is responsible for providing the reusable core assets that are exploited during application engineering when assembling or customizing individual applications [19]. Although there are other activities, such as product management, in this section we do not try to be exhaustive, but only discuss those activities directly related to this paper and relevant to our approach. Thus, following the nomenclature used in [19], the activities, usually performed iteratively and in parallel, of domain engineering that correlate with our approach are:

The *Domain Requirements Engineering* activity describes the requirements of the complete family of products, highlighting both the common and variable features across the family. In this activity, commonality analysis is of great importance for aiding in determining which are the common features and which of them are present only in some products. The models used in this activity for specifying features show when a feature is optional, mandatory or alternative in the family. One of the most accepted models here is *feature models* [4]. A feature is a characteristic of the system that is observable by the end user [7]. Features represent a concept quite similar to system goals (used in AOSE) and the models used to represent them present a correlation with hierarchical system goal requirement documents [16]. Our approach is based on this correlation.

In Figure 1, we show a subset of the feature model from our case study. As shown, in this kind of model the features for all products are shown along with information on whether they are mandatory, optional, or alternative. For example, the feature *flight and orbit* is mandatory, while the feature *walk* is optional. In addition, the features *snake*, *amoeba*, etc. must be present only if their parent is present, and, as they are related by an or-relation, when a product possesses the feature *walk* it must also possess at least one of the former features.

The *Domain Design* activity produces architecture-independent models that define the features of the family and the domain of application. Many approaches have been discussed in the literature to perform this modeling. Some of these approaches use role models to represent the interfaces and interactions needed to cover certain functionality independently (a feature or a set of features). The most representative are [6,21], but similar approaches have appeared in the OO field, for example [5,20]. We build on this correlation using agent-based role models at the acquaintance organization to represent features independently.

Then, in the *Domain Realization* activity, a detailed architecture of the family is produced adding mechanisms such as components that can be customized, or frameworks for these components, in order to enable the rapid derivation of products. In SPL, there exist some approaches where collaboration-based models (role models) are composed to produce the core architecture, e.g. [6,21]. In these approaches, component-based models are used where each component is assigned a set of interfaces and a set of connectors to specify interactions among them. Again, this is similar approach to the approach of some AOSE methodologies in building the architecture, called the *structural organization*, e.g. [22].

3.2 Overview of MaCMAS/UML

The organizational metaphor has been proven to be one of the most appropriate tools for engineering a MAS, and has been successfully applied, e.g., [10,12,22]. It shows that a MAS organization can be observed from two viewpoints [22]:

Acquaintance point of view: shows the organization as the set of interaction relationships between the roles played by agents.

Structural point of view: shows agents as artifacts that belong to sub-organizations, groups, teams. In this view agents are also structured into hierarchical structures showing the social structure of the system.

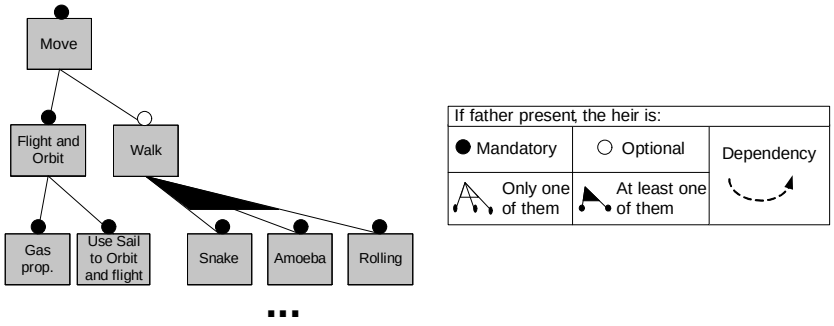

Fig. 1. Sub-set of the feature model of our case study

Both views are intimately related, but they show the organization from radically different viewpoints. Since any structural organization must include interactions between agents in order to function, it is safe to say that the acquaintance organization is always contained in the structural organization. Therefore, a natural map is formed between the acquaintance organization and the corresponding structural organization. This is the process of assigning roles to agents [22]. Then, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [8].

MaCMAS is the AOSE methodology that we use for our approach and is based on previously developed concepts [18][2]. It is specially tailored to model complex acquaintance organizations [17].

We have adopted this approach because it presents several common features with SPL approaches, that eases the integration of both fields. Going into details, the main reasons are: First, after applying it we obtain a hierarchical diagram, the traceability diagram, that is quite close to a feature model. Second, it matches well with product lines, since it also produces a set of role models that represent the materialization of each system goal at the analysis level. Third, it provides UML-based models which are the de-facto standard in modeling, and which will decrease the learning-curve for engineers. Fourth, it provides techniques for composing acquaintance models, which is needed for building the structural organization of the system, allowing us to group together those features that are common to all of the products in the product line and thus, build the core architecture.

For the purposes of this paper we only need to know a few features of MaCMAS, mainly the models it uses. Although a process for building these models is also needed, we do not address this in this paper, and refer the interested reader to the literature on this methodology. From the models it provides, we are interested in the following:

[2] See http://james.eii.us.es/MaCMAS/ for details and case studies of this methodology

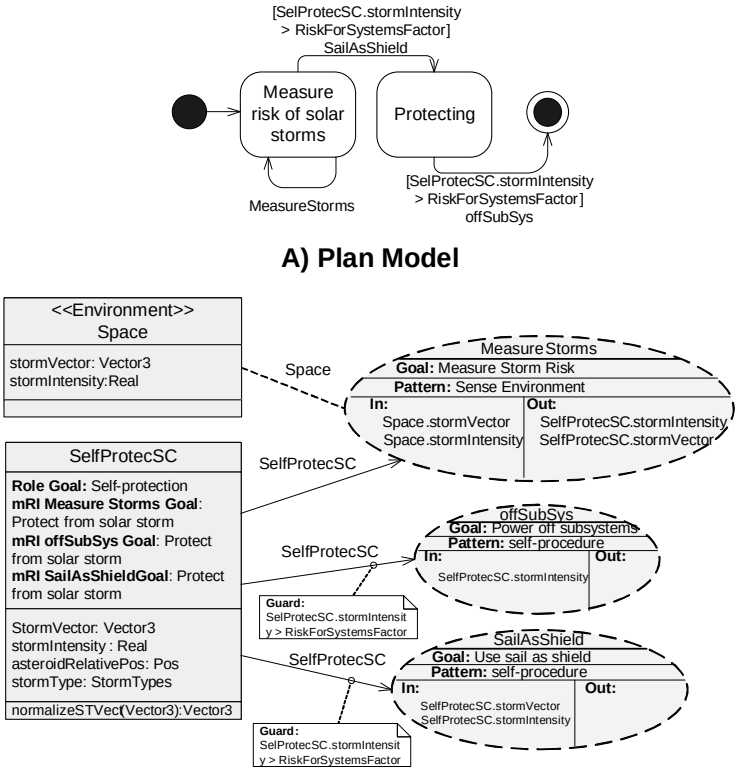

Fig. 2. "Self-protection from solar storms" autonomic property model

a) Static Acquaintance Organization View: This shows the static interaction relationships between roles in the system and the knowledge processed by them. In this category, we can find models for representing the ontology managed by agents, models for representing their dependencies, and role models. For the purposes of this paper we only need to detail role models:

Role Models: show an acquaintance sub-organization as a set of roles collaborating by means of several *multi-Role Interactions* (mRI) [14]. mRIs are used to abstract the acquaintance relationships amongst roles in the system. As mRIs allow abstract representation of interactions, we can use these models at whatever level of abstraction we desire.

In Figure 2.B), we show the role model corresponding to an autonomic feature of our case study that models how to materialize protection from a solar storm at the domain design level. Roles are represented as UML-interface-like shapes, and mRIs are shown as UML-collaboration-like shapes. Both notations are extended with some information required for modeling agents, such as goals, or collaboration patterns. One example

of role a is *SelfProtectSC*; it shows its goals, the knowledge that should be managed to fulfill these goals, and the services it provides to be able to achieve its goals. One example of an mRI is *Measure Storms*: it is linked to its participant roles, and it shows the goal it fulfills, the pattern of collaboration between its participating roles, and the knowledge it both needs and produces in order to fulfill the goal.

b) Behavior of Acquaintance Organization View: The behavioral aspect of an organization shows the sequencing of mRIs in a particular role model. It is represented by two equivalent models:

Plan of a role: separately represents the plan of each role in a role model showing how the mRIs of the role sequence. It is represented using UML 2.0 ProtocolStateMachines [11]. It is used to focus on a certain role, while ignoring others.

Plan of a role model: represents the order of mRIs in a role model with a centralized description. It is represented using UML 2.0 StateMachines [11]. It is used to facilitate the understanding of the whole behavior of a sub-organization.

In Figure 2.A), we show the plan of the role model. As can be seen, each transition in the state machine represents an mRI execution. In this model, we can show that we have to execute the mRI *measure storms* until the risk of solar storms is higher than a constant, shown with a guard. Thus, when the guard holds, we have to execute the mRI *sailAsShield*.

c) Traceability view: This model shows how models in different abstraction layers relate. It shows how mRIs are abstracted, composed or decomposed by means of *classification*, *aggregation*, *generalization* or *redefinition*. Notice that we usually show only the relations between interactions because they are the focus of modeling, but all the elements that compose an mRI can also be related. Finally, since an mRI presents a direct correlation with system goals, traceability models clearly show how a certain requirement system goal is refined and materialized. Notice that we do not show this model since, adding commonalities and variabilities, it is equivalent to the feature model that we show later.

4 Overview of our approach for building the core architecture

From all the activities that have to be performed for setting up a product line, we show here a subset concerning the development of the core architecture from the modeling point of view. Thus, we do not cover activities such as product management since it falls out of the scope of this paper.

In Figure 3, we show the Software Process Engineering Metamodel (SPEM) definition of the software process of our approach. The first stage to be performed consist of developing a set of models in different layers of abstraction where we obtain a MaCMAS traceability model and a set of role models showing how each goal is materialized. This is achieved by applying the MaCMAS software

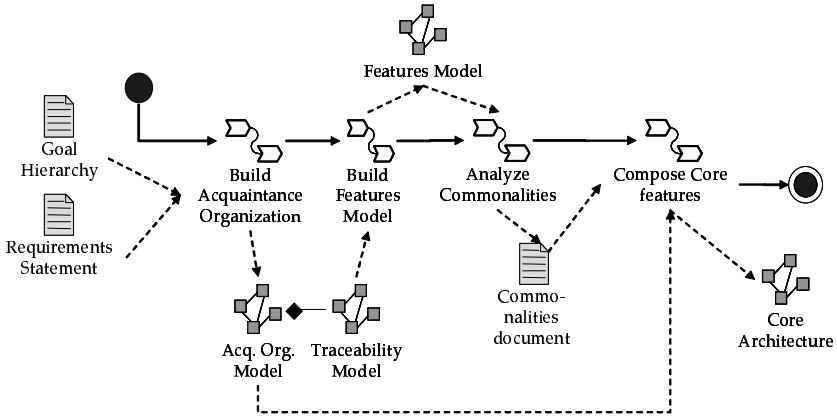

Fig. 3. Overview of our approach

process. The second activity shown is responsible for adding commonalities and variabilities to the traceability model. Later, we perform a commonality analysis to find out which features, called core features, are more used across products. Finally, we compose the role models corresponding to these features to obtain the core architecture. The following sections describe these activities.

5 Building the acquaintance organization and the feature model

After applying MaCMAS, as we were building a MAS that covers the functionality of all products in the family, we obtain a model of the acquaintance organization of the system: role models, plan models and a traceability model. Once we have built the acquaintance organization, we have to modify the traceability diagram to add information on variability and commonalities, as shown in Figure 5, to obtain a feature model of the family. We do not detail this process since it relies on taking each node of the traceability diagram and determining if it is mandatory, optional, alternative, or-exclusive, or if it depends on other(s), as shown in the figure.

MaCMAS guides this entire process using hierarchical goal-oriented requirement documents from which all of the models are produced. Thus, there is a direct traceability between system goals and role models. This traceability is feasible since when a system goal is complex enough to require more than one agent in order to be fulfilled, a group of agents are required to work together. Hence, a role model shows a set of agents, represented by the role they play, that join to achieve a certain system goal (whether by contention or cooperation). MaCMAS uses mRIs to represent all of the joint processes that are required and are carried out amongst roles in order to fulfill the system goal of the role

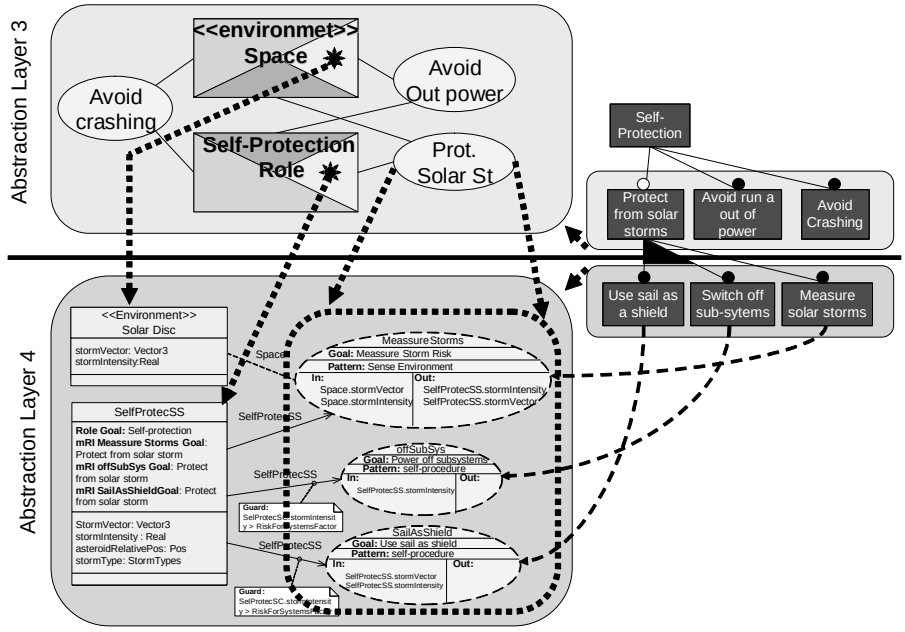

Fig. 4. Role model/features relationship

model. These also pursue system sub-goals as shown in Figure 4, where we can see the correlation between these elements and the feature model obtained from the traceability diagram. Note that the role model of this figure can be also seen in Figure 2.

6 Commonality analysis

To build the core architecture of the system we must include those features that appear in all the products and those whose probability of appearing in a product is high. In [1,2] the authors define the commonality of a feature as the percentage of products defined within a feature model that contains the feature. A calculation method for this and many other operations related to feature models analysis is proposed using Constraint Satisfaction Problems (CSP). The definition of commonality is the following:

Definition 1 (Commonality). *Let M be a feature model and F the feature within M whose commonality we want to calculate. Let P be the set of products defined by M and P_F the subset of products P containing F. commonality(F) is defined as follows:*

$$commonality(M, F) = \frac{|P_F| \cdot 100}{|P|}$$

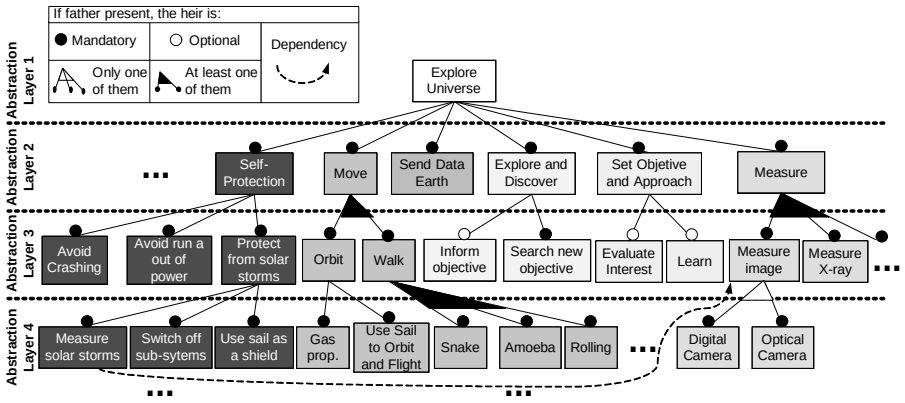

Fig. 5. Features model of our case study

Considering the previous definition, for any full-mandatory feature (this means a feature that appears in all the products in the family) $P_F = P$, its commonality will be 100%. For any other non-full-mandatory feature, $P_F \subset P$ and therefore its commonality will be less than 100%.

Calculating the commonality of every feature, we can easily determine which are the full-mandatory features and consequently the role models that must be used to build the core architecture. For those features whose commonality is less that 100%, we have to consider which of them will be part of the core and which will not. We propose to use a threshold, that must be calculated empirically for each domain, to make this decision. Consequently those features whose commonality is above the threshold will be also used to build the core architecture.

In addition, tools that help engineers with automated analysis of features models are of high value [1]. We have extended the prototype[3] presented in [1] to automatically calculate the commonality of all the features of our case study. The results obtained with the prototype are shown in Figure 6. As shown in this figure, these features are ordered by their commonality. The figure also show the threshold that we have selected, set up at the 60%, for considering a feature to be core or not.

We use the following fictitious scenario to document our example: We have realized that the commonality for the features *self-protection from a solar storm* and *orbiting* is 100%. Thus we have to add them to the core architecture, since they appear in all the possible products.

As these features are related, since if a spacecraft is orbiting and measuring and it determines that there exists a risk of a solar storm, the spacecraft must first escape the orbit and later power down subsystems or use its sail as a shield

[3] This prototype along with this and other case studies is available at http://www.tdg-seville.info/topics/spl

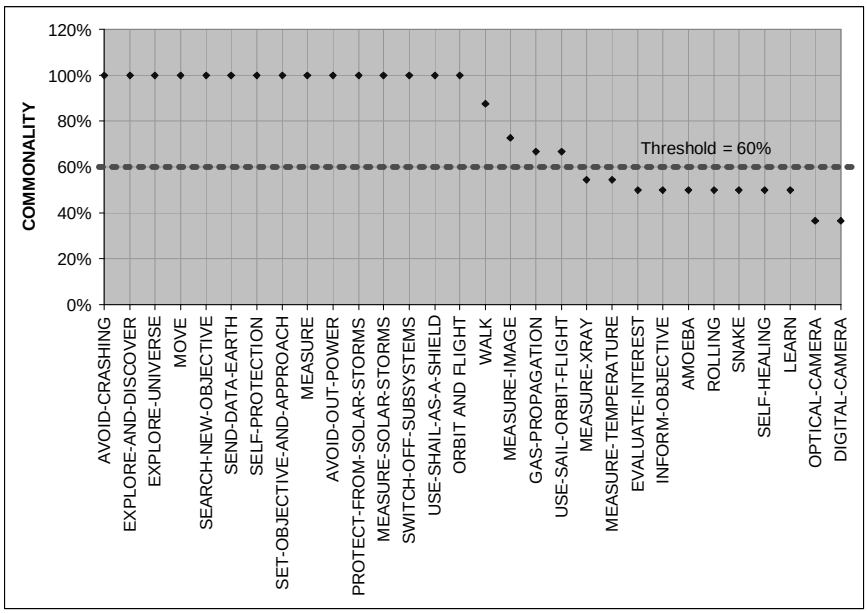

Fig. 6. Commonalities of the features in our example

to avoid crashing, we are forced to compose them to model their dependencies and provide agents with all the roles needed to safely protect from solar storms in any situation. Notice that we have limited our example to two role models to simplify the example, but in the real world we must also take into account the rest of the related features.

Once we have determined the set of features, and thus, the set of role models to be taken into account for the core architecture, we must compose them as described in the following section.

7 Composition of the core features

We have to take into account that when composing several role models, we can find: *emergent roles* and *mRIs*, artifacts that appear in the composition yet they do not belong to any of the initial role models; *composed roles*, the roles in the resultant models that represent several initial roles as a single element; and, *unchanged roles* and *mRIs*, those that are left unchanged and imported directly from the initial role models.

Once those role models to be used for the core architecture have been determined, we must complete the core architecture by composing role models. Importing an mRI or a role requires only adding it to the composite role model. The following shows how to compose roles and plans.

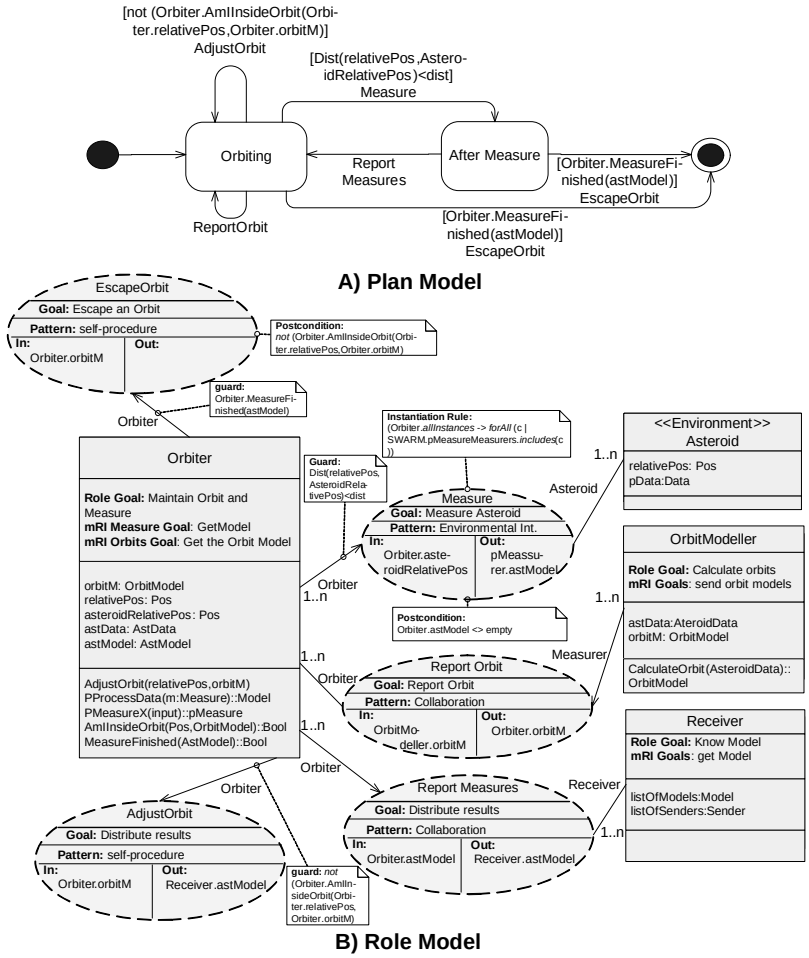

Fig. 7. "Orbiting and measuring an asteroid" autonomous property

7.1 Composing roles

When several roles are merged in a composite role model, their elements must be also merged as follows:

Goal of the role: The new goal of the role abstracts all the goals of the role to be composed. This information can be found in requirements hierarchical goal diagrams or we can add it as the *and* (conjunction) of the goals to be composed. In addition, the role goal for each mRI can be obtained from the goal of the initial roles for that mRI.

Cardinality of the role: It is the same as in the initial role for the corresponding mRI.

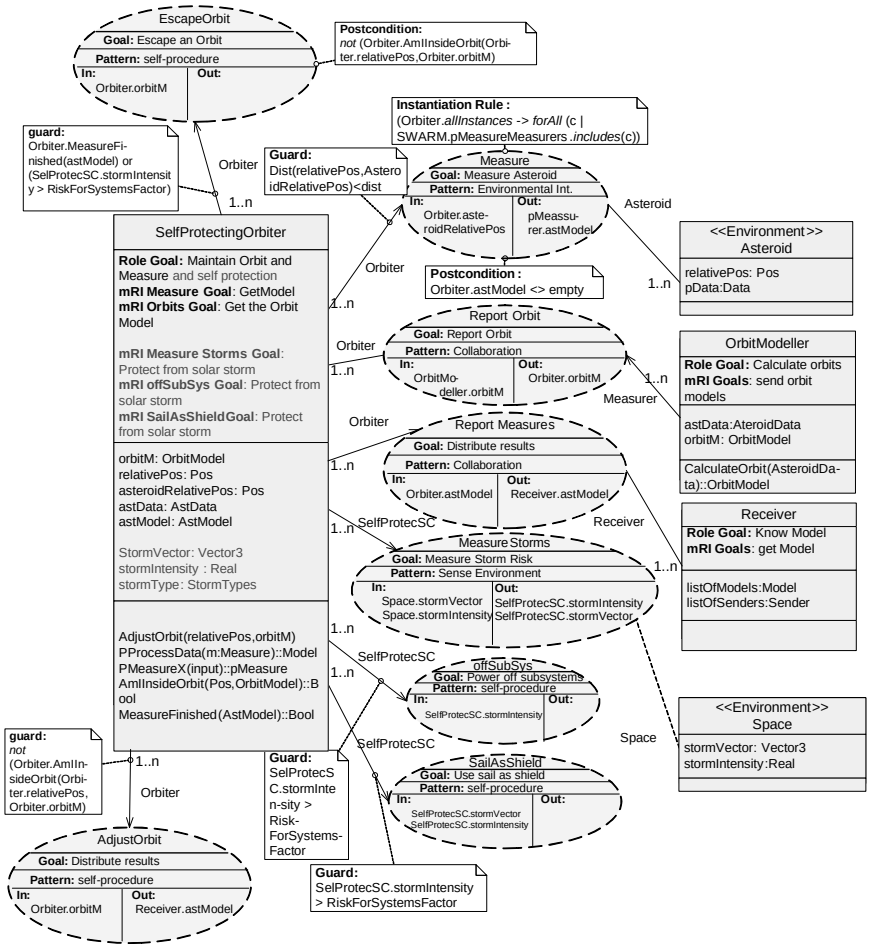

Fig. 8. Composed role model

Initiator(s) role(s): If mRI composition is not performed, as in our case study, this feature does not change.

Interface of a role: All elements in the interfaces of roles to be merged must be added to the composite interface. Note that there may be common services and knowledge in these interfaces. When this happens, they must be included only once in the composite interface, or renamed, depending on the composition of their ontologies.

Guard of a role/mRI: The new guards are the *and* (conjunction) of the corresponding guards in initial role models if roles composed participate in the same mRI. Otherwise, guards remain unchanged.

In our case study, we have to compose the role models corresponding to the features *self-protection from a solar storm* and *orbiting*. The model for the

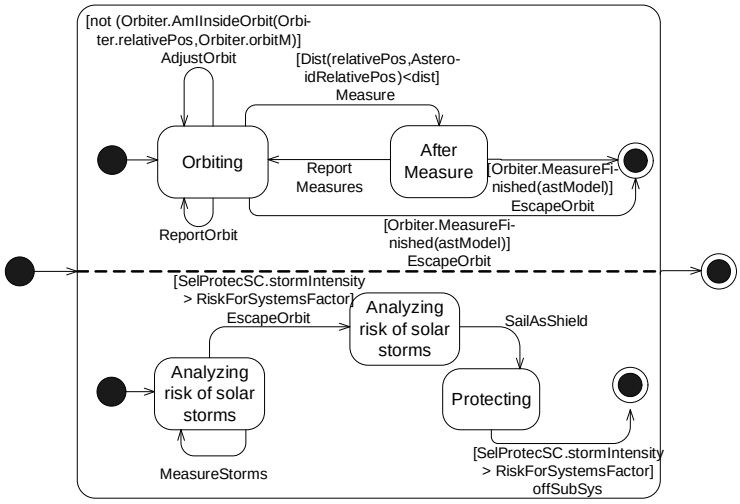

Fig. 9. Composed plan model

former feature was shown in Figure 2, and the role model of the later is shown in Figure 7.

After applying the approach described above, the composed role model obtained is shown in Figure 8. As we can see, the roles *Orbiter* and *SelfProtectSC* have been composed into a single role called *SelfProtectingOrbiter*. These roles has been composed because the agent that plays one of the roles also has to play the other as they are dependent (mRIs *escape orbit*, *power off sub-systems*, and *use sail as a shield* have to be sequenced in a certain way).

Since the rest of the roles are orthogonal, that is to say they do not interact with each other, they have been left unchanged and all mRIs have been also added without changes.

7.2 Composing plans

The composition of plans consists of setting the order of execution of mRIs in the composite model, using the role model plan or role plans. We provide several algorithms to assist in this task: extraction of a role plan from the role model plan and vice versa, and aggregation of several role plans; see [13] for further details of these algorithms.

Thanks to these algorithms, we can keep both plan views consistent automatically. Depending on the number of roles that have to be merged we can base the composition of the plan of the composite role model on the plan of roles or on the plan of the role model. Several types of plan composition can be used for role plans and for role model plans:

Sequential: The plan is executed atomically in sequence with others. The final state of each state machine is superimposed with the initial state of the

state machine that represents the plan that must be executed, except the initial plan that maintains the initial state unchanged and the final plan that maintains the final state unchanged.

Parallel: The plan of each model is executed in parallel. It can be documented by using concurrent orthogonal regions of state machines (cf. [11]).

Interleaving: To interleave several plans, we must build a new state machine where all mRIs in all plans are taken into account. Notice that we must usually preserve the order of execution of each plan to be composed. We can use algorithms to check behavior inheritance to ensure that this constraint is preserved, since to ensure this property the composed plan must inherit from all the initial plans [9].

The composition of role model plans has to be performed following one of the plan composition techniques described previously. Later, we are interested in the plan of one of the composed roles, as it is needed to assign the new plan to the composed roles; we can extract it using the algorithms mentioned previously.

We can also perform a composition of role plans following one of the techniques to compose plans described previously. Later, if we are interested in the plan of the composite role model, for example for testing, we can obtain it using the algorithms mentioned previously.

Regarding our example, as the self-protection must be taken into account during the whole process of orbiting and measuring, and not in a concrete state, we must perform a parallel composition of their plans, with a minor interleaving of the mRI *escape orbit* in the self protection plan, as is shown in Figure 9.

8 Conclusions

The field of software product lines offers many advantages to organizations producing a range of similar software systems. Reported benefits of the approach include reduced time-to-market, reduced costs, and reduced complexity. Simultaneously, the ability to spread development costs over a range of products has enabled adopters to invest more significantly in software quality.

Multiagent systems have a wide field of applicability, across a whole plethora of domains. However, many key features, including communication, planning, replication, security mechanisms, to name but a few, are likely to be very similar across all MAS, particularly in a given domain.

Key to the development of MAS-PLs is the identification of the core MAS from which a family of concrete products may be derived. We have described an initial approach to building this part of the infrastructure needed to enable a product line approach in MAS.

The approach matches well with existing AOSE methodologies and promises to open a field of research and development that may make MAS and MAS-based systems more practical in an industrial context.

We are continuing to investigate the use of such an approach in current and future NASA missions. For example, we have applied MAS-PL to manage evolutionary systems that benefits from the results of this paper [15]. Initial results

are promising and over time we envisage significant benefits from employing a product line approach to such missions.

References

1. D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.
2. D. Benavides, A. Ruiz-Cortés, P. Trinidad, and S. Segura. A survey on the automated analyses of feature models. *XV Jornadas de Ingeniería del Software y Bases de Datos,JISBD 2006*, 2006.
3. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, Aug. 2001.
4. K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison–Wesley, 2000.
5. D. D'Souza and A. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison–Wesley, Reading, Mass., 1999.
6. A. Jansen, R. Smedinga, J. Gurp, and J. Bosch. First class feature abstractions for product derivation. *IEE Proceedings - Software*, 151(4):187–198, 2004.
7. K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie-Mellon University, November 1990.
8. E. A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, Apr./June 2000.
9. B. Liskov and J. M. Wing. Specifications and their use in defining subtypes. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 16–28. ACM Press, 1993.
10. J. Odell, H. Parunak, and M. Fleischer. The role of roles in designing effective agent organisations. In A. Garcia and C. L. F. Z. A. O. J. Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems*, number 2603 in LNCS, pages 27–28, Berlin, 2003. Springer–Verlag.
11. O. M. G. (OMG). Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. www.omg.org.
12. H. V. D. Parunak and J. Odell. Representing social structures in UML. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 100–101, Montreal, Canada, 2001. ACM Press.
13. J. Peña, R. Corchuelo, and J. L. Arjona. Towards Interaction Protocol Operations for Large Multi-agent Systems. In *Proceedings of FAABS'02*, volume 2699 of *LNAI*, pages 79–91, MD, USA, 2002. Springer–Verlag.
14. J. Peña, R. Corchuelo, and J. L. Arjona. A top down approach for mas protocol descriptions. In *ACM Symposium on Applied Computing SAC'03*, pages 45–49, Melbourne, Florida, USA, 2003. ACM Press.
15. J. Peña, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash. Managing the evolution of an enterprise architecture using a mas-product-line approach. In *5th International Workshop on System/Software Architectures (IWSSA'06)*, page to be published, Nevada, USA, 2006. CSREA Press.
16. J. Peña, M. G. Hinchey, and A. Ruíz-Cortes. Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, December 2006.

17. J. Peña, R. Levy, and R. Corchuelo. Towards clarifying the importance of interactions in agent-oriented software engineering. *International Iberoamerican Journal of AI*, 9(25):19–28, 2005.
18. J. Pena. *On Improving The Modelling Of Complex Acquaintance Organisations Of Agents. A Method Fragment For The Analysis Phase*. PhD thesis, University of Seville, 2005.
19. K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering : Foundations, Principles and Techniques*. Springer, September 2005.
20. T. Reenskaug. *Working with Objects: The OOram Software Engineering Method*. Manning Publications, 1996.
21. Y. Smaragdakis and D. Batory. Mixin layers: an object–oriented implementation technique for refinements and collaboration-based designs. *ACM Trans. Softw. Eng. Methodol.*, 11(2):215–255, 2002.
22. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), September 2003.