# Building the Core Architecture of a Multiagent System Product Line: With an example from a future NASA Mission

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

[1] University of Seville, Spain
joaquinp@us.es, aruiz@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 core architecture for 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 AOSE methodologies are concerned with developing a single Multiagent System. We propose an initial approach 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 and Motivation

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 tends. 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 core architecture from which customized products can be rapidly generated, reducing time-to-market, costs, etc. [2], 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 [15], 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).

One of the first steps 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 fragment for analysing Complex Multiagent Systems (MaCMAS) [17], an AOSE methodology fragment 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.

We use goal-oriented requirement documents, role models, and traceability diagrams in order to build a first model of the system. Later, we use information on variability and commonalities throughout the products to propose a transformation of the prior models that represent the core architecture of the family.

## 2 Background on product lines and related work

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 in 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 [5]. Entering into details, we might say that, generally, both stages can be further divided into requirements, analysis, design, and implementation (a typical software development lifecycle).

The *domain requirements* phase describes the requirements of the complete family of products, highlighting both the common and variable features across the family. In this phase, commonality analysis is of great importance for aiding in determining which are the commonalities and variabilities. The models used in this phase for specifying features show when a feature is optional, mandatory or alternative in the family. One of the most accepted models here are *feature models* [3]. A feature is a characteristic of the system that is observable by the end user [7]. As can be seen, features represent a concept quite similar to system goals and the models used to represent them present a correlation with hierarchical system goals requirement documents [15]. Our approach is based on this correlation.

The *domain analysis* phase 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 [6,19], but similar approaches have appeared in the OO field, for example [4,18]. We build on this correlation using agent-based role models at the acquaintance organization to represent features independently.

Then, in the *domain design* phase, a core 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, role models are composed to produce the core architecture. Here, 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 structural organization, e.g. [20].

## 3 Preliminaries

Before presenting our approach, we must show the main concepts needed to contextualize it.

### 3.1 Acquaintance Organization *vs.* Structural Organization

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., [10,12,20]. It shows that a MAS organization can be observed from two viewpoints [20]:

**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.

Both views are intimately related, but they show the organization from radically different viewpoints. Since any structural organization must include interactions between their 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 [20]. Then, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [8].

### 3.2 Overview of MaCMAS/UML

MaCMAS is the AOSE methodology fragment that we use for our approach [17][1]. It is specially tailored to model complex acquaintance organizations [16].

We have adopted this approach for several reasons. 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 to compose acquaintance models, which is needed for building the structural organization of the system, allowing us to group such system goals that are common for all the products in the product line, an thus helping us to build the feature model.

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 fragment. From the models it provides, we are interested in the following:

[1] See www.tdg-seville.info/members/joaquinp/macmas/ for details and case studies of this methodology

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 Interaction* (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.

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, p. 422]. 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, p. 446]. It is used to facilitate easy understanding of the whole behavior of a sub-organization.

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.

## 4 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,[2] 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.

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

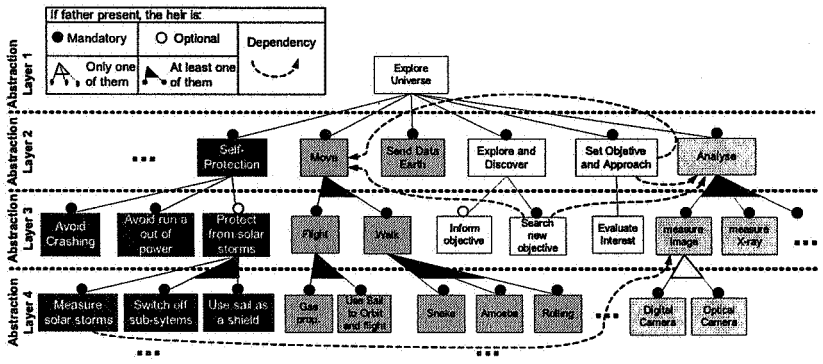


**Fig. 1.** Features model of our case study

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 in space forming swarms, sub-swarms and teams, and deployed to study asteroids of interest in the asteroid belt. Saturn Autonomous Ring Array (SARA) is also a future mission similar to PAM whose goal is analyzing 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 storms, while in SARA as a higher gravitational force exists, and there is no risk of solar storms, the spacecraft should be capable of avoiding collisions with particles of the rings and with other spacecraft, without caring about solar storms.

ANTS represents a number of sub-missions, each with common features, but with a wide range of applicability, and hence several products. It lends itself naturally to an SPL approach.

## 5 From requirements to traceability diagrams, its 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 the information on variability and commonalities, as shown in Figure 1, to obtain a feature model of the family. We do not detail this process since it relies only 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.

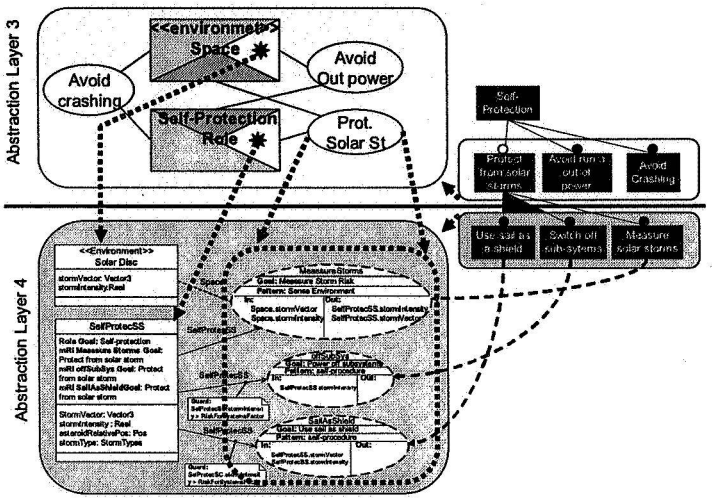


**Fig. 2.** Role model/features relationship

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. When a system goal is complex enough to require more than one agent in order to be fulfilled, it requires a group of agents 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 represents all required joint processes that are carried out amongst roles to fulfil the system goal of the role model using mRIs, which also pursue system sub-goals as shown in Figure 2, 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 3.

## 6 Our approach for building the core architecture

To build the core architecture of the system we must include those features (that are linked with role models) that are common for all products and whose probability of appearing in a product is high. Once we have determined these features, we must compose their role models to build the structural organization of our core architecture. In the following, we detail how to select the features and how to compose them.

### 6.1 Commonality analysis

To perform commonality analysis, that is to say, to obtain the probability that a feature is common to all products in the family, we use the definition given in [1], which proposed a transformation of the feature model into a Constraint Satisfaction Problem (CSP) over which we apply the following definition for commonality:

**Definition 1 (Commonality).** *Let $\psi_M$ be a CSP representing a feature model and $F$ the feature we want to know its commonality.*

$$commonality(\psi_M, F) = \frac{cardinal(filter(\psi_M, F = true))}{cardinal(\psi_M)}$$

Where $cardinal(filter(\psi_M, F = true))$ gives us the number of products that contain the feature and $cardinal(\psi_M)$ gives us the total number of products, deriving the probability of a feature being present in a product.

When we derive a 1, this means that this feature, and consequently its role model, must be used to build the core architecture. When we obtain a value less than 1, we take into account only those features for which we obtain a commonality greater than a threshold that must be determined empirically for each domain.

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.

### 6.2 Composing role models

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 and mRIs*, the roles and mRIs in the resultant models that represent several initial roles or mRIs 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.

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 is a new goal that abstracts all the role 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.

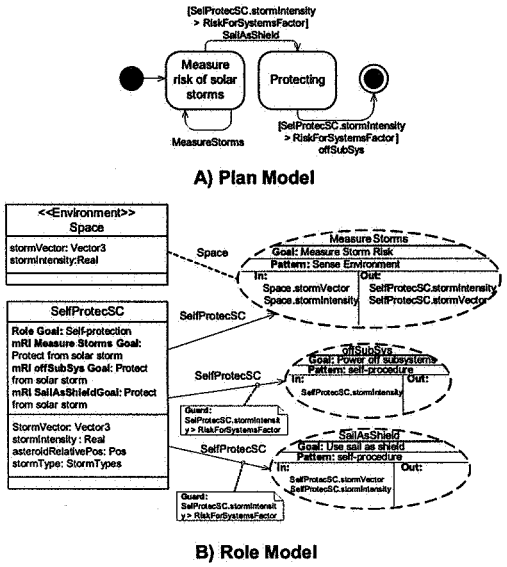


**Fig. 3.** *Self-protection from solar storms* autonomic property model

**Initiator(s) role(s):** If mRI composition is not performed, as in our case, 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. Notice 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.

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

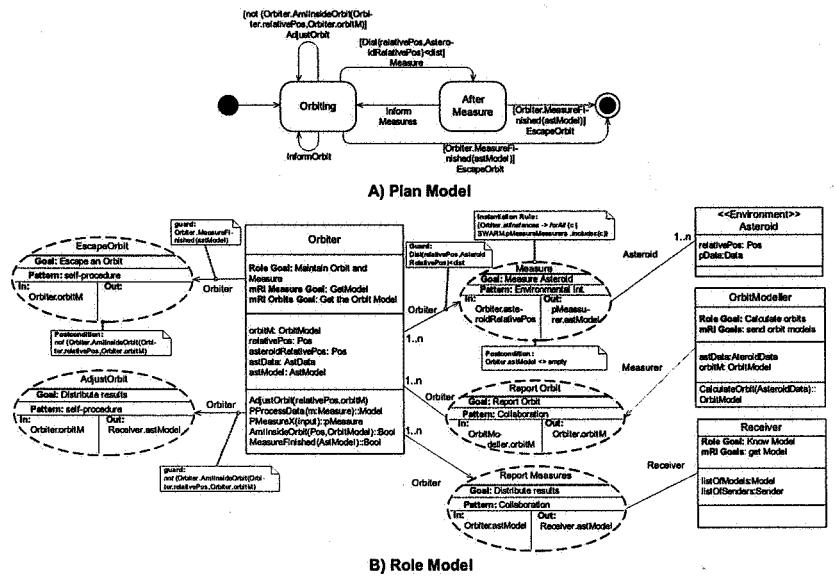


**Fig. 4.** *Orbiting and measuring an asteroid* autonomous property

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, p. 435]).

**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.

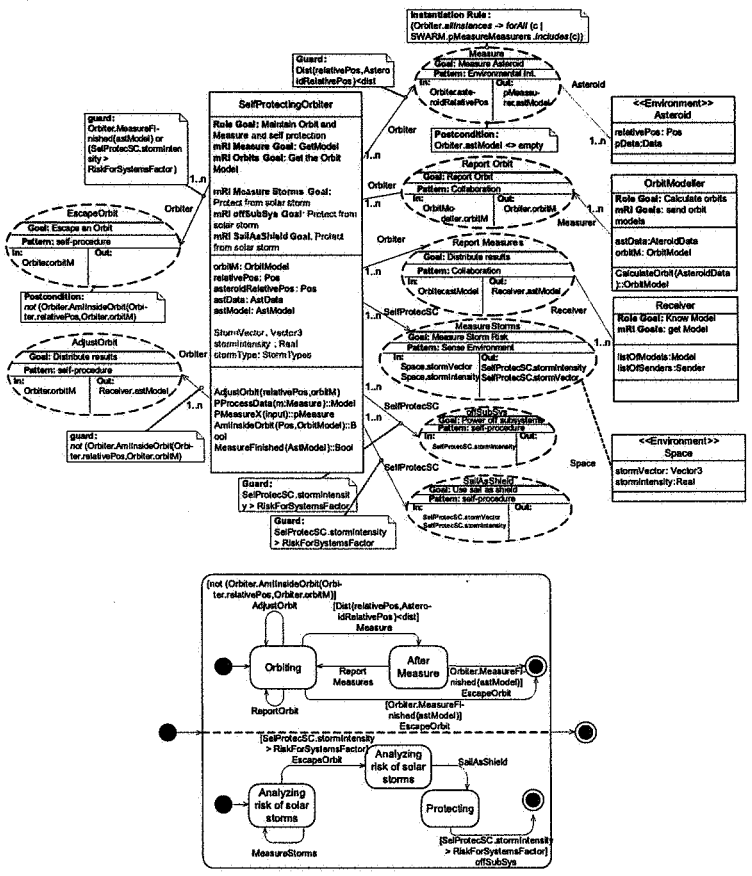





**Fig. 5.** Composed Role Model and plan model

## 7 Example of how to build part of the core architecture of our case study

We use the following fictitious scenario to document our example: We have realized that the commonality for the features for self-protection from a solar storm, whose role model is shown in Figure 3, and for orbiting, whose role model is shown in Figure 4, is sufficient to add them to the core architecture, since they appear in all the possible flying missions.

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 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.

As a result, we must compose both models and their plans. The composition of both role models is shown in Figure 5. As we can see, the roles *Orbiter* and *SelfProtectSC* have been composed into a single role called *SelfProtectingOrbiter* following the prescription shown in the previous section. We can observe that the rest of roles have been left unchanged and that all mRIs have been also added without changes.

In addition, 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, as is shown in Figure 5.

## 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. 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. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* SEI Series in Software Engineering. Addison–Wesley, Aug. 2001.
3. K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Addison–Wesley, 2000.

4. D. D'Souza and A. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach.* Addison–Wesley, Reading, Mass., 1999.
5. M. Harsu. A survey on domain engineering. Technical Report 31, Institute of Software Systems, Tampere University of Technology, December 2002.
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 and M. G. Hinchey. Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, December 2006. Submitted and pre-accepted.
16. J. Peña, R. Levy, and R. Corchuelo. Towards clarifying the importance of interactions in agent-oriented software engineering. *International Iberoamerican Journal of AI*, (25):19–28, 2005.
17. 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.
18. T. Reenskaug. *Working with Objects: The OOram Software Engineering Method.* Manning Publications, 1996.
19. 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.
20. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), September 2003.